CSCE 221 Cover Page Ryan Wenham / 627002098 / ryanjw0903@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	sam		
Web pages (provide URL)			
Printed material	your notes		
Other Sources			

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 Ryan Wenham Date

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

```
#include <cstdlib>
#include <list>
#include <iostream>
using namespace std;
void func(list<int> one, list<int> two){
   list<int> result;
   int i = 0;     int j = 0;
   while(i<one.size() && j<two.size()){
      if(one[i] < two[j]){
       i++;
     } else if(one[i]>two[j]){
               j++;
     }else{
     result.push_back(one[i]);
     i++;  j++;
     }
   }
   for(int k = 0; k<result.size(); k++){
     cout << result[k] << " ";
   }
}
int main(){
   vector<int> one {1,2,3,5,9};
   vector<int> two {2,3,7,8,9,10};
   func(one,two);
}
```

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

      i. The run time is O(one+two)

   (c)

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.

```
int count(Node * n){
   if(n == nullptr){
    return 0;
   }
   else {
    return 1+count(n->next)
   }
}
```

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

      i. T(1) = 0

      ii. T(n) = T(n-1) + 1

(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 + 1 + ... 1(n-1)

   ii. O(n-1)

(d)

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.

```
int fdmax(vector<int> v, int n){
   if(n==1){
     return v[0];
   } else{
     int temp = fdmax(v, n-1);
     if(temp > v[n]){
       return temp;
     }
     return v[n];
     }
}
Used same main as part One
```

   (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) = c + c ... c(n-1)

      ii. O(n-1)

   (d)

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

   (a) Provide recurrence relations and their solutions.

      i. Best Case: T(n) = 2T(n/2)+n / O(n)

      ii. Average Case: T(n) = (n+1)T(n-1) + 2n / O(n logn)

      iii. Worst Case: T(n) = T(n-1) + n / O(nˆ2)

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

      i. Best Case: 1,4,2,5,9,6,12 , Start piovt at value that partions with balnces sides so the median value be best, that will make the pivot 5

      ii. Avergae Case: 1,5,6, 2,9,4 Start pivot at end 4.

      iii. Worst Case: 1,4,19,23,27 Start pivot at 1

   (c)

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.

```
int BinaryNode::size(BinaryNode *t) {
   if (t == nullptr)
      return 0;
    if(t->left != nullptr && t->right != nullptr)
      return 1 + size(t->left) + size(t->right);
    if(t->left != nullptr && t->right == nullptr)
       return 0 + size(t->left)
    if(t->left == nullptr && t->right != nullptr)
       return 0 + size(t->right)
    else
     return 0;
}
O(n)
```

(a)