CSCE 221 Cover Page

First NameAbdullah Last Name Ahmad UIN 927009064

User Name Abdullah2808 E-mail address Abdullah2808@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			Article
People			
Web pages (provide URL)	https://www.geeksforgeeks.org/quick-sort/ https://bit.ly/37suao8	https://bit.ly/3d0jOwJ Wiki Article (URL shortened)	
Printed material			
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 Abdullah Ahmad Date 6/12/2020

Homework 2

due June 10 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

```
list<int> intersection(list<int> L1, list<int> L2)  {
        // construct a new list to return
        list<int> L3;
        // iterators
        list<int>::iterator it1;
        list<int>::iterator it2;
        it1 = L1.begin();
        it2 = L2.begin();
        while (it1 != L1.end() && it2 != L2.end())
{
                if (*it1 < *it2)
                        it1++;
                else if (*it2 < *it1)
                        it2++;
                else {
                        L3.push_back(*it1);
                        it1++;
                        it2++;
                }
        }
        return L3;
}

void print(list<int> L1)
{       for (auto it = L1.begin(); it != L1.end(); ++it)
                cout << ' ' << *it; }

int main() {
        list<int> L1, L2, L3;
        for (int i = 0; i < 9; i++) {
                L1.push_back(i);
        }
        for (int i = 0; i < 15; i++) {
                L2.push_back(i);        }
        L3 = intersection(L1, L2);
        print(L1);      cout << endl;   print(L2);      cout << endl;   print(L3);
        return 0; }
```

```
0 1 2 3 4 5 6 7 8
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
0 1 2 3 4 5 6 7 8
```

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

   The running time of the algorithm in Big-O notation is O(n).

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 num = 0;struct Node {      int value;        struct Node* next;};void Insert(struct Node** head, int valu
```

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

The recurrence relation is $T(n) = T(n-1) + C$

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

$$
\begin{aligned}
T(n-1) &= T(n-2) + C' \\
T(n) &= T(n-1) + 2C' \\
T(n-2) &= T(n-3) + C; \\
T(n) &= T(n-3) + C; \\
T(n) &= T(n-x) + xC; \\
T(n) &= T(n-n) + nC'
\end{aligned}
$$

This means that if $T(n) = T(0) + nC', T(0) = C$ then $T(n) = C + nC'$. So the Big-O notation is 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.

```
template <class T> const T& max(const T& a, const T& b) {
        return (a < b) ? b : a;
    }
int findMax( int A[], int n )
{
        if (n == 1)
                return A[0];
        return max(A[n - 1], findMax(A, n - 1));
}
// test 1
int main() {
        int A[] = { 1, 35,  3 , 4, 15, 25, 12 };
        int n = 8;
        cout << "The␣max␣element␣is:␣" << findMax(A, n) << endl;
        return 0;
}

"The␣max␣element␣is:␣35"

//
int main() {
        int A[] = { 1, 12,  45 , -12, 15, 25, 12 };
        int n = 8;
        cout << "The␣max␣element␣is:␣" << findMax(A, n) << endl;
        return 0;
}

"The␣max␣element␣is:␣45"
```

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

The recurrent relation is $T(n) = T(n-1) + 1$

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

$$\begin{aligned} T(n) &= T(n-1) + 1 \\ &= T(n-2) + 1 + 1 \\ &= 1 + 1 + ...n \\ &= O(n) \end{aligned}$$

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

   (a) Provide recurrence relations and their solutions.

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

   Best Case: $O(nlog(n))$

   The recurrence relation is $T(n) = 2T(n/2 + O(n)$ and can be solved using the master method to obtain the above Big-O runtime. This case occurs when the parition selects the element in the middle as the pivot.

   Worst Case: $O(n^2)$

   The recurrence relation is $T(n) = T(n-1) + O(n)$, and can be solved using the tree method or iterative method to obtain the above Big-O runtime. This case occurs when either the largest or smallest element is selected as the pivot.

   Average Case: $O(nlog(n))$

   The recurrence relation is dependent on how the data is partitioned we assume $T(n) = T(n/9) + T(9n/10) + O(n)$ if split n/9 in one set and 9n/10 in another set. The recurrence relation can be solved using the tree method and the Big-O is given above. This case occurs whenever the pivot point is randomly selected

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.

```
#include <iostream>
#include<queue>
using namespace std;
struct Node
{
        int data;
        struct Node* left, * right;
};
unsigned int CountOfNodesBothChildren(struct Node* node)
{
        if (!node)
                return 0;
        queue<Node *> q;
        int count = 0;
        q.push(node);

        while (!q.empty())
        {
                struct Node* temp = q.front();
                q.pop();
                if (temp->left && temp->right)
                        count++;
```

```
                if(temp->left != NULL)
                        q.push(temp->left);
                if (temp->right != NULL)
                        q.push(temp->right);
        }       return count; }

struct Node* newNode(int data)
{       struct Node* node = new Node;
        node->data = data;
        node->left = node->right = NULL;
        return (node); }

int main() {
        struct Node* root = newNode(6);
        root->left = newNode(13);
        root->right = newNode(2);
        root->left->right = newNode(13);
        root->left->left = newNode(43);
        root->right->left = newNode(46);
        root->right->right = newNode(36);
        root->right->left->left = newNode(11);
        root->left->right->right = newNode(17);
        root->right->right->left = newNode(15);
        root->left->left->right = newNode(123);

        cout << CountOfNodesBothChildren(root);
        return 0; }

output: 3
```

The time complexity is in Big-O notation is $O(n)$