

CSCE 221 Cover Page

First Name Rong Last Name Xu UIN 928009312

User Name	Abby-xu	E-mail address	abby.xu915@gmail.com
-----------	---------	----------------	----------------------

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)			
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 Rong Xu Date 2020/10/10

Homework 2

due October 16 at 11:59 pm to eCampus

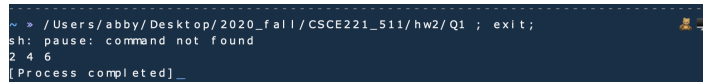
1. (20 points) Given two sorted lists, L1 and L2, write an efficient C++ code to compute $L1 \cap L2$ using only the basic STL list operations.

(a) Provide evidence of testing: submit your code

```
#include <iostream>
#include <list>
using namespace std;

list<int> inter_list(list<int> L1,
list<int> L2) {
    if(L1.empty() || L2.empty())
        std::exit(-1);
    L1.merge(L2);
    list<int> return_list;
    for(list<int>::iterator it = L1.begin(); it != L1.end(); it++) {
        if(*it == *--it)
            return_list.push_back(*it);
        ++it;
    }
    return return_list;
}

int main() {
    int A1[]={1,2,3,4,5,6};
    int A2[]={2,4,6,8,9,10};
    list<int> iL1(A1, A1+6);
    list<int> iL2(A2, A2+6);
    list<int> iL3 = inter_list(iL1, iL2);
    list<int>::iterator it = iL3.begin();
    while(it != iL3.end()) {
        cout << *it++ << " ";
    }
    system("pause");
    return 0;
}
```



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

- i. From the official website, the running time of function `std::merge()` is $O(n)$.
 - A. <https://en.cppreference.com/w/cpp/algorithm/merge>
 - B. n is the number of elements of List 1.
- ii. The big-o of for loop is $O(n+k)$.
 - A. n is the number of elements of List 1
 - B. k is the number of elements of List 2
 - C. In this case, the list L1 was merged by List 2. Thus the List 1 now has $(n + k)$ element.
- iii. Base of the statements above, the running time is $O(n) + O(n + k) = O(n + k)$

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

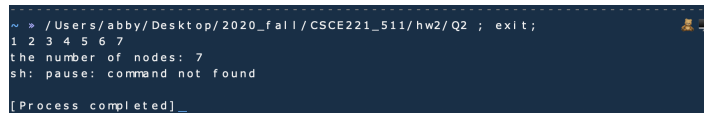
- (a) Test your function using different singly linked lists. Include your code.

```
#include<iostream> using namespace std;
class Node {
public:
    int data;
    Node *next;
    Node(int da = 0, Node *p = NULL) {
        this -> data = da;
        this -> next = p;
    }
};

class List{
private:
    Node *head,*tail; public:
    List(){head = tail = NULL;};
    ~List(){delete head; delete tail;};
    void print() {
        Node *p = head;
        while (p != NULL) {
            cout << p -> data << "\a";
            p = p->next;
        }
        cout << endl;
    };
    void Insert(int da) {
        if (head == NULL) {
            head = tail = new Node(da);
            head -> next = NULL;
            tail -> next = NULL;
        } else {
            Node *p = new Node(da);
            tail -> next = p;
            tail = p;
            tail -> next = NULL;
        }
    };
    Node* first() {return head;}
};

int count_node(Node* n) {
    if(n == NULL)
        return 0;
    else
        return 1 + count_node(n -> next);
}

int main() {
    List l1;
    l1.Insert(1);l1.Insert(2);l1.Insert(3);l1.Insert(4);
    l1.Insert(5);l1.Insert(6);l1.Insert(7);
    l1.print();
    cout << "the number of nodes:" << count_node(l1.first()) << endl;
    system("pause");
    return 0;
}
```



```
~ > /Users/abby/Desktop/2020_fall/CSCE221_S11/hw2/Q2 ; exit;
1 2 3 4 5 6 7
the number of nodes: 7
sh: pause: command not found
[Process completed]
```

- (b) Write a recurrence relation that represents your algorithm.
- i. Base case: When n is a null pointer, return 0.
 - A. $T(0) = c_1$ for some constant c_1
 - ii. Recursive case: When n is not a null pointer.
 - A. $T(n) = c_2 + T(n - 1)$ for some constant c_2
- (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. If we knew $T(n - 1)$, we could solve $T(n)$.
 - ii. $T(n) = T(n-1) + c_2 = T(n-2) + c_2 + c_2 = T(n-2) + 2c_2 = T(n-3) + 3c_2 = \dots = T(n-k) + kc_2$
 - iii. So we have $T(n) = T(n - k) + k * c_2$ for all k
 - iv. If we set $k = n$, we have $T(n) = T(n - n) + nc_2 = T(0) + nc_2 = c_1 + nc_2 = O(n)$

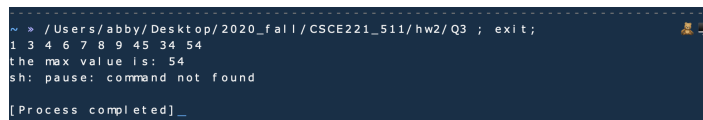
3. (20 points) Write a C++ recursive function that finds the maximum value in an array (or vector) of integers *without* using any loops.

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

```
#include <iostream>

int find_max(int A[], int n) {
    if(n == 1)
        return A[0];
    return std::max(A[n - 1], find_max(A, n - 1));
}

int main() {
    int A[] = {1,3,4,6,7,8,9,45,34,54};
    int n = sizeof(A) / sizeof(A[0]);
    for (int i = 0; i < 10; i++)
        std::cout << A[i] << " ";
    std::cout << "\nthe max value is: " << find_max(A, n) << std::endl;
    std::system("pause");
    return 0;
}
```



```
~ > /Users/abby/Desktop/2020_fall/CSCE221_511/hw2/Q3 ; exit;
1 3 4 6 7 8 9 45 34 54
the max value is: 54
sh: pause: command not found
[Process completed]
```

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

- i. Base case: When n reach to the first element in the array/vector.
 - A. $T(0) = c1$ for some constant $c1$
- ii. Recursive case: Fund the max element when not reach to the first element..
 - A. $T(n) = c2 + T(n - 1)$ for some constant $c2$

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

- i. If we knew $T(n - 1)$, we could solve $T(n)$.
- ii. $T(n) = T(n-1) + c2 = T(n-2) + c2 + c2 = T(n-2) + 2c2 = T(n-3) + 3c2 = \dots = T(n-k) + kc2$
- iii. So we have $T(n) = T(n - k) + k * c2$ for all k
- iv. If we set $k = n$, we have $T(n) = T(n - n) + nc2 = T(0) + nc2 = c1 + nc2 = O(n)$

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

(a) Provide recurrence relations and their solutions.

i. Best case:

A. Recurrence Relation: $T(n) = T(n/2) + T(n/2) + O(n)$ and $T(1) = 0$

B. Solve it by iteration method:

$$\begin{aligned} T(n) &= 2T(n/2) + O(n) \\ &= 2(2T(n/4) + O(n/2)) + O(n/2) + O(n) \\ &= 4T(n/4) + 2 * O(n) \\ &= \dots \\ &= 2^k T(n/2^k) + K * O(n) \\ &= O(n \log_2 n) \end{aligned}$$

ii. Worst case:

A. Recurrence Relation: $T(n) = T(n - 1) + T(1) + n = T(n - 1) + n$ and $T(1) = 0$

B. Solve it by iteration method:

$$\begin{aligned}
 T(n) &= T(n-1) + n \\
 &= T(n-2) + (n-1) + n \\
 &= T(n-3) + (n-2) + (n-1) + n \\
 &= \dots \\
 &= T(n-k+1) + (n-k+2) + \dots + (n-1) + n \\
 &= \dots \\
 &= T(1) + 2 + 3 + \dots + (n-1) + n \\
 &= \frac{n(n+1)}{2} - 1 \\
 &= O(n^2)
 \end{aligned}$$

iii. Average case:

A. Recurrence Relation: $T(n) = T(cn) + T((1-c)n) + n$ and $T(1) = 0$

B. Solve it by iteration method: Let h_L be the height of left subtree and h_R be the height of right subtree. Notice that $h_R > h_L$.

C. solve for h_L and h_R :

$$\begin{aligned}
 c^{h_L} &= 1/n \\
 h_L &= -\log_c h \\
 h_L &= \log_{1/c} h
 \end{aligned}$$

$$\begin{aligned}
 (1-c)^{h_R} &= 1/n \\
 h_R &= -\log_{(1-c)} h \\
 h_R &= \log_{1/(1-c)} h
 \end{aligned}$$

D. The big-o is $O(n \log_2 n)$

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

- i. For the best case: the input has already been sorted and the pivot is just the middlest element in the list.
- ii. For the average case: we usually choose the pivot in the middle or just choose the random index of the pivot.
- iii. For the worst case: the input list is reversed and we start to choose the pivot at the beginning of the list until the last one.

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 <vector>
using namespace std;

std::vector<int> nodes;
struct BiTNode {
    int data;
    struct BiTNode* lchild;
    struct BiTNode* rchild;
};

void create_tree(BiTNode* &tree) {
    int data; cin >> data;
    if (data != '\n') {
        if (data == -1) {
            tree = nullptr;
        } else {
            tree = new BiTNode;
            tree->data = data;
            create_tree(tree->lchild);
            create_tree(tree->rchild);
        }
    }
}

void pre_order_traverse(BiTNode* &tree) {
    if (tree) {
        cout << tree->data << " ";
        pre_order_traverse(tree->lchild);
        pre_order_traverse(tree->rchild);
    }
}

void count_node(BiTNode* &tree) {
    cout << "data: " << tree->data;
    if (tree->lchild != nullptr && tree->rchild != nullptr)
        nodes.push_back(tree->data);
    if (tree->lchild == nullptr && tree->rchild == nullptr)
        return;
    count_node(tree->lchild);
    count_node(tree->rchild);
}

int main() {
    BiTNode* T;
    create_tree(T);
    /*      input here is:      2 3 5 -1 -1 6 -1 -1 4 -1 -1
           2
          / \
         3   4
        / \
       5   6
      */
    pre_order_traverse(T); //2, 3, 5, 6, 4
    count_node(T);
    cout << "\nthe nodes are:" << endl;
    for (int j = 0; j < nodes.size(); j++)
        cout << nodes[j] << " ";
    cout << "\nthe number of node with two children is:" << nodes.size() << endl;
    system("pause");
    return 0;
}
```

```
> /Users/abby/Desktop/2020_fall/CSCE221_511/hw2/Q5 ; exit;
2
3
5
6
4
the nodes are:
2 3 5 6 4
the number of node with two children is: 2
sh: pause: command not found
[Process completed]
```

(a) I think the the big-o if my algorithm is $O(\log_2 n)$