National University of Computer and Emerging Sciences, Lahore Campus

Course:	Data Structures	Course Code:	CS 201,218
Program:	BS(Computer Science)	Semester:	Fall 2019
Duration:	60 Minutes	Total Marks:	25
Paper Date:	25-Sept-2019	Page(s):	5
Section:	ALL	Section:	
Exam:	Midterm Exam 1	Roll No:	

Instruction/Notes: Answer in the space provided
You can ask for rough sheets but they will not be graded or marked
In case of confusion or ambiguity make a reasonable assumption. Questions are not allowed

Good luck!

Question 1: (Marks: 5)

Perform a step count analysis on the code fragment given below and derive an equation for T(n) for the worst case. Also compute the big-oh (tight bound). Show complete working

```
Void Function(int *arr, int n)
{
        for (int k = 0; k<n; k++)
                arr[k] = 0;
        for (int p = 2; p*p <= n; p++) {
                for (int i = 1; i<= n; i += 2)
                        arr[i]++;

        }
}
```

Solution

k=0	1
k<n	n+1
k++	2n // k=k+1
arr[k]=0	n
p=2	1
p*p<=n	$\sqrt{n}+1$
p++	$\sqrt{n}$
i=1	$\sqrt{n}$
i<=n	$\sqrt{n}\left(\frac{n}{2}+1\right)$
i+=2	$2\sqrt{n}\,\frac{n}{2}$
arr[i]++	$2\sqrt{n}\,\frac{n}{2}$

$T(n) = 1+n+1+2n+n+\sqrt{n}+1+\sqrt{n}+\sqrt{n}+\frac{n^{3/2}}{2}+\sqrt{n}+n^{3/2}$

$+n^{3/2}$

$= \frac{5n^{3/2}}{2}+3n+4\sqrt{n}+3$

$O(n^{3/2})$

5

Roll Number: _____ Section: _____

Question 2: (Marks: 5)

Consider the following function of class singly linked list with head pointer only. Describe the functionality of ***mystery(const T & d)*** in 2-3 lines.

```
bool mystery(const T & d){
        node<T> * t1,*t2 ;
        t1 = head;
        while(t1!=nullptr && t1->data!=d){
                t2 = t1;
                t1 = t1->next;
        }
        if(t1!=nullptr){
                if(t1!= head){
                        t2->next = t1->next;
                        t1->next = head;
                        head = t1;
                }
                return true;
        }
        return false;
}
```

Solution

If any node has data equal to 'd' then it places it at the head, if it is not already the head. If it is already the head then it does nothing and if it does not exist or the list is empty then it does nothing.

Question 3: (Marks: 7+ 8)

You are given a sorted singly linked list that may contain duplicate elements. Your task is to remove the duplicate elements in the list and count the number of occurrences/frequency of unique elements. For example you are given a list L below:

1 -> 1 -> 3-> 3 ->3 -> 3 -> 5 -> 9 -> 9 -> 12 -> 15 -> 15 -> 15 -> 15 -> 15 -> 15 -> 17 ->

L after duplicate removal must be
1 -> 3 -> 5 -> 9 -> 12 -> 15 -> 17 ->
Also list of frequencies will be
2 -> 4 -> 1 -> 2 -> 1 -> 6 -> 1->
Where 2 is frequency of 1 and 4 is frequency of 3 and so on.

Write an ***efficient*** C++ function ***removeDuplicates()*** that take a singly linked list ***L*** and remove all the duplicate values in ***L***. This function should also return a singly linked list ***F*** of integers that contain the frequencies of unique elements initially present in the list ***L***.

Note: Less credit will be awarded to less efficient solutions.

Roll Number: [redacted] Section: [redacted]

For Section (A, B, C, D)

removeDuplicate() is not a member function of the List class. So use iterators to process the lists L and F. You can assume that List and iterator classes are already implemented. List has following members:

```
template <class T>
class List{
        node<T> * head;
        node<T> *tail;
public:
        class iterator{
        node<T>* current;
        public:
                iterator();//initialize current to nullptr
                iterator& operator++();//moves current to next node
                T& operator*(); returns data of current node
                bool operator!=(iterator & rhs); //return true if this and rhs points to same node
                friend class List<T>;
        };
        iterator begin(); //returns an iterator that points to start of the list
        iterator end(); //return an iterator that points to end of the list
        List(); //initialize head and tail pointers to nullptr
        ~list(); //delete all the nodes in the list
        void insertAtEnd(const T&); // O(1) function that inserts d at the end of the list
};
```

First write the appropriate ***remove()*** function in the List class using iterators that remove the node pointed by the iterator in the list and then use it in ***removeDuplicates()***.

void remove (Iterator obj)

For Section(E and R)

removeDuplicates() is not a member function of the class List. So use the current pointer to remove node pointed by current pointer. You can assume that List class is already implemented. List has following members:

```
template <class T>
class List{
        node<T> * head;
        node<T> *tail;
        node<T> *current;
        public:
        List();//initialize head and tail pointers to nullptr
        ~list(); //delete all the nodes in the list
        void insertAtEnd(const T&); // O(1) function that inserts d at the end of the list
};
```

First write the appropriate ***remove()*** function in the List class that remove the node pointed by the current in the List and then use that function in ***removeDuplicates()***.

Roll Number: _____ Section: _____

Write C++ code for remove() function

```
void remove (List<T>::Iterator obj)
{
    Node<T>* ptr = head;
    if (obj.current == ptr)
    {
        Node<T>* temp = head;
        head = head->next;
        delete[] temp;
    }
    else
    {
        while (ptr->next != obj.current)
            ptr = ptr->next;
        Node<T>* temp = ptr->next;
        ptr->next = ptr->next->next;
        delete[] temp;
    }
}
```

At the end

Write C++ code for removeDuplicates() function

```
List<T> removeDuplicates(List<T>& biglist)
{
    List<T> frequency;
    for(Iterator ptr1 = biglist.begin(); ptr1 != biglist.end();
                                          ++ptr1)
    {
        T mydata = *ptr1;
        int freq = 1;
        List<T>::Iterator ptr2 = ptr1;
        ++ptr2;
```

```
while (*ptr2 == mydata)
{
list<T>::Iterator temp = ptr2;
   ++ptr2;
biglist.remove(temp);
    Freq++;
}
frequency.insertAtend(freq);
}
}
```

send prev to avoid this = remove - 1

7

```
void remove(list<T>::Iterator obj)
{
    list<T>::Iterator curr = begin();
    list<T>::Iterator prev = nullptr;
    while (curr != obj)
    {
        prev = curr;
        ++curr;
    }
    if (prev.current == nullptr)
    {
        Node<T>* temp = head;
        head = head->next;
        delete[] temp;
        if (head == nullptr)
            tail = nullptr;
    }
    else
    // next page
```

```
{
    if (tail == cuss.current)
        tail = prev.current;

    Node<T>* temp = cuss.current;

    prev.current->next = cuss.current->next;

    delete [] temp;
    obj.current = nullptr;
}
}
```