COSC 2436
Practice Exam

SECTION	Question	Value	Score
Q1: Linked List – 35 Points	I	8	
	II	8	
	III	3	
	IV	8	
	V	8	
Q2: Recursion – 25 Points	I	10	
	II	10	
	III	5	
Q3:Big O – 20 Points	I	3	
	II	4	
	III	7	
	IV	6	
Q4: Sorting - 20 Points	I	5	
	II	10	
	III	5	
	Total	**100**	

Question 1: Linked List - 35 Points

I. Write a Function to RECURSIVLEY count the number of nodes in a singly Linked list? This function should return the total number of nodes in the lsit
(8 points)

```
struct Node {
  int val;
  Node* next;
};

Int countNodes(Node* head) {
      If(head == nullptr)
            return 0;
      return 1 + countNodes(head->next);
}

};
```

II. Given a double-linked list, write a function that prints nodes that are "peak". A node is considered "peak" if the one before and after are less than the node. Desired output for given examples: (**8 points**)

Desired output:

a. [1,2,5,4] -> 5
b. [1,2,6,3,7,5] -> 6,7
c. [1,2,3] -> No output

```
struct Node {
  int data;
  Node* prev;
  Node* next;
};

void peak(Node* head) { // write code below

    Node* cu = head->next;
    while(cu->next != nullptr) {
        if(cu->prev->data < cu->data && cu->data > cu->next->data)
            cout << cu->data << " ";
        cu = cu->next;
}
```

III. What is the output of the printList function? (3 points)

```
struct Node {
  int val;
  Node* next;
};

class LinkedList{
  private:
    Node* head;
  public:
    LinkedList() {
      head = NULL;
    }
    void myFunc(int num) {
      Node *myNode = new Node;
      myNode->val = num;
      if(head == NULL) {
        head = myNode;
      }
      else if(num % 2 == 0) {
        myNode->next = head;
        head = myNode;
      } else {
        Node *curr = head;
        while(curr->next != NULL) {
          curr = curr->next;
        }
        curr->next = myNode;
      }
    }
    void printList() {
      Node* curr = head;
      while(curr != NULL) {
        cout << curr->val << endl;
        curr = curr->next;
      }
    }
};

int main() {
  int nums[] = {1,2,4,13,10,7,9,12,6,8};
  LinkedList myList = LinkedList();
  for(int i = 0; i < 10; i++) {
    myList.myFunc(nums[i]);
  }
  myList.printList();
}
```

8 6 12 10 4 2 1 13 7 9

IV. Given a circular linked list, write a function that prints out the next greatest value for each node. Print -1 if you're looking at the greatest node already, meaning there is no greater node to print. Your code can print the simplified output; the extended output is just for your understanding. **(8 points)**

Example: input: 6,3,10,9,1

output: node 6: next greatest value = 9
node 3: next greatest value = 6
node 10: next greatest value = -1
node 9: next greatest value = 10
node 1: next greatest value = 3

simplified output: 9,6,-1,10,3

```
     struct Node {
       int data;
       Node* prev;
       Node* next;
     };

void nextGreatestValue(Node* head) { // write code below

    Node* cu = head;
    vector<int> v;
    do {
        Node* temp = cu->next;
       int nextValue = -1;
        while(temp != cu) {
           if(temp->data > cu->data && nextValue = -1)
             nextValue = temp->data;
          else if(temp->data > cu->data && temp->data <
nextValue)
             nextValue = temp->data;
        }
        v.push_back(nextValue);
        cu = cu->next;
    }while(cu != head);

    for(int i=0; i<v.size(); i++) {
        cu->data = v[i];
        cu = cu->next;
    }
}
```

V. Write a function addTwoNumbers that accepts 2 doubly linked lists as input(the lists already created). It then adds the values from the nodes. Returns the head of the new list. Each node value will be an integer [0,9]. **(8 points)**

For example to add 387 + 214 = 601 will be
List 1 : 3 ⇋ 8 ⇋ 7

List 2 : 2 ⇋ 1 ⇋ 4

Result : 6 ⇋ 0 ⇋ 1

```
 struct Node {
   int data;
   Node* next;
 };

Node* addTwoNumbers(Node* tail_1, Node* tail_2) {

    int sum = 0;
    int carry = 0;
    linkedlist result;
    while(tail1 != nullptr || tail2 != nullptr || carry != 0)
{
        sum = 0;
        if(tail1 != nullptr) {
            sum += tail1->data;
            tail1 = tail1->prev;
        }
        if(tail2 != nullptr) {
            sum += tail2->data;
            tail2 = tail2->prev;
        }
        sum += carry;
        carry = sum/10;
        result.addAtBeg(sum%10);
    }
    return result.getHead();
}
```

Question 2: Recursion - 25 Points

I. Write a RECURSIVE function that gets the factorial of the number n that's passed into the function. The factorial is the product of all numbers leading up to n. For example 4! (4 factorial) = 4 * 3 * 2 * 1 = 24? **(3 points)**

```
int factorial(int n)
{
    if (n == 0)
        return 1;
    return n * factorial(n - 1);
}
```

II. Consider a palindrome an array of numbers, such as 76967 or 566665. Write a recursive function **palindrome** that returns true if the array is a palindrome and false if it is not **(10 points)**

```
bool isPalindrome(int* arr, int low, int high)
{
     if (low >= high)
          return true;
     if (arr[low] != arr[high])
          return false;
     return isPalindrome(arr, low + 1, high - 1);
}
```

III. What is the output **(5 pts)**

```
#include<iostream.h>
void fun(int x)
{
  if(x > 1)
  {
     fun(--x);
     cout << --x <<" ";
     fun(x-1);
  }
}

int main()
{
  int a = 5;
  fun(a);
  return 0;
}
```

0 1 2 3 0

Question 3: Big O - 20 Points

I. What is the time complexity of the following code **(3 points)**

```
for(int i = 0; i < 10000000; i++) {
      for(int j = 0; j < n; i++) {
            cout << "Test" << endl;
        }
}
```

O(10000000*n) = O(n)

Even though this is a nested loop, the outer for loop have the time complexity of O(10000000) and this is a constant can be rewritten as O(1)

II. What is the time complexity of the following code **(4 points)**

```
void myFunc(int n, int m) {
  if(n <= 0) return;
  for(int i = 0; i < m; i++) {
    myFunc(n-1,m);
  }
}
```

O(m^n)

For each recursive call, we repeat the for loop of O(m), the recursive will be called for O(n) times

⇨ O(m) * O(m) * O(m) * ... (multiplies n times)

- In recursion we will have exponential time complexity,

  The base (the number that is being multiplied by itself) will depend on how many times you call the recursive function.

  Ex: In recursive fibonacci you call recursion function twice (fib(n-2) + fib(n-1)), hence fibonacci have a time complexity of O(2^n)

  If you have a recursive function that call itself 3 times, the time complexity will be O(3^n) and so on

III. Give the time complexities for the following two functions and explain which one has a shorter time complexity. **(7 points)**

```
void func_a(int n) {
  int i = 0;
  int j = 0;
  while(i < n) {
    while(j < n) {
      j = j * j;
    }
    j = 0;
    i += 1;
  }
 }
```

O(n*sqrt(n))

```
void func_b(int n) {
  int i = 0;
  int j = n;
  while(i < n) {
    while(j > 0) {
      j = j - 2;
    }
    j = 0;
    i += 1;
  }
```

O(n^2)

IV. What is the best Time Case time complexity for Selection Sort. Describe what the list must look like for it to have this time complexity. **(6 points)**

Accepted answer: O(n) or O(n^2)

List must be sorted already to have the best time complexity

Question 4: Sorting – 20 Points

I. Use Bubble Sort to sort the given array in ascending order. Write out the array for every iteration the sort occurs until the sorting is complete. Arr = {8,6,9,1,2,3}. DON'T Write a function just list the steps for every swap MANUALLY **(5 points)**

8,6,9,1,2,3

6,8,1,2,3,9

6,1,2,3,8,9

1,2,3,6,8,9

II. Given the array {24,31,12,17,9,15}. Implement a sorting function that would produce these arrays after each swap. State what type of sorting is used

(10 points)

Given array : {24,31,12,17,9,15}
1st swap : {9,31,12,17,24,15}
2nd swap : {9,12,31,17,24,15}
3rd swap : {9,12,15,17,24,31}

This is selection sort, you should implement selection sort here

III. Use Insertion Sort to sort the given array in descending order. Write out the array for every iteration the sort occurs until the sorting is complete. Arr = {1,5,9,17,2,6}. DON'T Write a function just list the steps for every swap MANUALLY

(5 points)

1,5,9,17,2,6

17,1,5,9,2,6

17,9,1,5,2,6

17,9,6,1,5,2

17,9,6,5,1,2

17,9,6,5,2,1