Instructions: This homework assignment covers some of the later course topics: Recursion, Stacks, Queues and Algorithm Analysis. The answers may be determined by reading the CS 1704 course notes and experimenting with code.

Opscan forms will be collected at the Friday class meeting on August 2. Be sure to select only one response per question; if you mark multiple answers you will receive no credit for that question. No late opscans will be accepted. Mutilated opscans may be discarded.

Consider the following member function as an addition to the LinkList & LinkNode classes from the course notes:

```cpp
// Counts the data elements contained in a LinkList object,
// Pre: the LinkList object is properly constructed;
//      the Curr pointer is at the Head of the LinkList prior to
//      the first call;
// Post: any data elements in the LinkList object have been
//       counted and the size of the list returned
//
int LinkList::SizeList() { // line 1
    if (__________) return(0); // line 2: stop recursion

    __________; // line 3

    return __________; // line 4: recurse
}
```

1. In order for the recursion to stop properly, how should the blank in line 2 be filled?

   1) moreList()  
   2) ! moreList()  
   3) isEmpty()  
   4) ! isEmpty()  
   5) this -> moreList()  
   6) None of these

2. In order to prepare for the recursive call in line 4, how should the blank in line 3 be filled?

   1) Curr = Curr->Next;  
   2) Head = Head->Next;  
   3) Curr = Curr->getNext();  
   4) Head = Head->getNext();  
   5) this.Advance();  
   6) None of these

3. In order for the recursion to be invoked correctly how should the blank in line 4 be filled?

   1) Curr->getNext()->SizeList() + 1  
   2) Head->getNext()->SizeList() + 1  
   3) this->SizeList() + 1  
   4) Curr->SizeList() + 1  
   5) Head->SizeList() + 1  
   6) None of these

4. How many times does the base case code in the above function execute for any given call?

   1) once at most  
   2) at least twice  
   3) as many times as the recursive code  
   4) equals the number of recursive calls  
   5) never, there is no need  
   6) None of the above
Consider the linked list class and list node declarations given below:

```cpp
class ItemType {
private:
    int Value;
public:
    ItemType();
    ItemType(int newValue);
    void setValue(int newValue);
    int getValue() const {return Value;} // Exclude one line
};
class LinkNode {
private:
    ItemType Data;
    LinkNode *Next, *Prev;
public:
    LinkNode();
    LinkNode(ItemType newData);
    bool setNext(LinkNode* newNext);
    bool setPrev(LinkNode* newPrev);
    bool setData(ItemType newData);
    ItemType getData() const;
    LinkNode* getNext() const;
    LinkNode* getPrev() const;
};
LinkNode *Head, *T;
```

Assume that the member functions above have been implemented correctly to carry out their intended task. Given the initial list structure:

![List Diagram]

5. What is the value returned by the call `cout << huh(Head, T);` to the following recursive function:

```cpp
int huh(LinkNode *R, LinkNode *S) {
    if ((R == NULL) || (S == NULL)) return 0;
    else if (R == S) return (R->getData()->getValue());
    else {
        if ((R->getNext() == S) && (S->getPrev() == R))
            return (R->getData()->getValue() + S->getData()->getValue());
        else
            return (R->getData()->getValue() + S->getData()->getValue() +
                    huh(R->getNext(), S->getPrev()));
    }
}
```

1) 13 
2) 23 
3) 32 
4) 33 
5) 42 
6) 55 
7) 111 
8) 143 
9) 0 

6. The previous function is an example of what type of recursive problem solving strategy?

1) going up (tail) recursion 
2) going down (head) recursion 
3) middle decomposition recursion 
4) edges & center decomposition 
5) backtracking 
6) None of the above
For the next two questions, consider the following recursive power function definition:

```c
// Returns value of x raised to the power exp.
// Pre: x is a positive real number
// exp is a nonnegative exponent
double Power2(double x, int exp) {
    if (exp < 0 || x <= 0.0) return -1.0; // logical error
    if (exp == 0) return 1.0;
    if (exp == 1) return x;
    double root = Power2(x, exp/2);
    if (exp % 2 == 1)
        return (x * root * root);
    else
        return (root * root);
}
```

Suppose the function is called as: `Power2(2.0, 27);

7. How many multiplications will be performed?

1) 1  
2) 3  
3) 6  
4) 7  
5) 15 
6) 18 
7) 19 
8) 20 
9) 0  
10) None of these

8. How many function calls (including the first) will be performed?

1) 1 
2) 5 
3) 10 
4) 12 
5) 15 
6) 18 
7) 19 
8) 20 
9) 0  
10) None of these

For the next four questions, apply the rules given in the course notes and in class to determine the most appropriate (tightest) big-O classification of the given function. All logarithms are base 2.

9. \( f(n) = 4n(2\log n - 6n^2) \) is \( \mathcal{O}(\underline{\text{}}) \)

1) \( \mathcal{O}(1) \)  
2) \( \mathcal{O}(\log n) \)  
3) \( \mathcal{O}(n) \)  
4) \( \mathcal{O}(n \log n) \)  
5) \( \mathcal{O}(n^2) \)  
6) \( \mathcal{O}(n^2 \log n) \)  
7) \( \mathcal{O}(2^n) \)  
8) None of these

10. \( f(n) = 7n + 3\log n \) is \( \mathcal{O}(\underline{\text{}}) \)

1) \( \mathcal{O}(1) \)  
2) \( \mathcal{O}(\log n) \)  
3) \( \mathcal{O}(n) \)  
4) \( \mathcal{O}(n \log n) \)  
5) \( \mathcal{O}(n^2) \)  
6) \( \mathcal{O}(n^2 \log n) \)  
7) \( \mathcal{O}(2^n) \)  
8) None of these
11. \( f(n) = \log(8 + 4n^2) \) is \( O( \underline{\text{______}} ) \) // tricky

1) \( O(1) \)  
2) \( O(\log n) \)  
3) \( O(n) \)  
4) \( O(n \log n) \)  
5) \( O(n^2) \)  
6) \( O(n^2 \log n) \)  
7) \( O(2^n) \)  
8) None of these

12. \( f(n) = 3n^2 + 5\log n + 9n \) is \( O(\underline{\text{______}}) \)

1) \( O(1) \)  
2) \( O(\log n) \)  
3) \( O(n) \)  
4) \( O(n \log n) \)  
5) \( O(n^2) \)  
6) \( O(n^2 \log n) \)  
7) \( O(2^n) \)  
8) None of these

---

For the next three questions, assume that \( N \) is an integer variable with a positive value, and any other declarations necessary to make the given code fragment syntactically and logically correct.

13. Execution of the following code fragment is \( O(\underline{\text{__________}}) \).

```c
int sum = A[0];
for (int Idx = 1; Idx < N; Idx++) {
    sum += A[Idx];
}
```

1) \( O(1) \)  
2) \( O(\log N) \)  
3) \( O(N) \)  
4) \( O(N \log N) \)  
5) \( O(N^2) \)  
6) None of these

14. Execution of the following code fragment is \( O(\underline{\text{__________}}) \).

```c
int i = 1, Total = 0;
while (i < N + 1) {
    int Sum = 0;
    for (int j = N; j >= 0; j--)
        Sum++;
    Total += Sum;
    i++;
}
```

1) \( O(1) \)  
2) \( O(\log N) \)  
3) \( O(N) \)  
4) \( O(N \log N) \)  
5) \( O(N^2) \)  
6) None of these
15. Execution of the following code fragment is $O(\text{__________})$.

```c
int i = 1, Total = 0;
while (i < N + 1) {
  int Sum = 0;
  for (int j = N; j >= i; j--)
    Sum++;
  Total += Sum;
  i++;
}
```

1) $O(1)$  
2) $O(\log N)$  
3) $O(N)$  
4) $O(N \log N)$  
5) $O(N^2)$  
6) None of these

16. Execution of the following code fragment is $O(\text{__________})$.

```c
int S = 0;
int M = N;
int T = M + M;
S += M * M;
```

1) $O(1)$  
2) $O(\log N)$  
3) $O(N)$  
4) $O(N \log N)$  
5) $O(N^2)$  
6) None of these
For questions 17 through 20, we consider a non-recursive version of the `intComma()` function from the course notes. The recursion is eliminated by making use of an internal `Deque` (double-ended) object, assuming a Dequeue class described in the course notes, with `string` typedef'd to `ItemType`. (See the supporting course notes regarding string streams.)

```c++
void intComma(long num) {
    if (num < 0) {
        cout << '-';
        num = -num;
    }
    Deque Parts;
    while (num >= 1000) {
        long tail = num / 1000; // Line 1
        stringstream Out(" ");
        if (tail < 10)
            Out << "00";
        else if (tail < 100)
            Out << "0";
        Out << tail;
        num = num / 1000; // Line 2
    }
    cout << setw(3) << num;
    string Next;
    while (!Parts.Empty()) {
        // Line 3
        cout << ',' << Next;
    }
}
```

17. How should the blank in Line 1 be filled?

1) num / 1000  
2) num % 1000  
3) num - 1000  
4) num  
5) None of these

18. How should the blank in Line 2 be filled?

1) `EnqRear(Out.str())`  
2) `Parts[Top] = Out`  
3) `Parts.EnqRear(Out.str())`  
4) `Parts.DeqRear(Out.str())`  
5) None of these

19. How should the blank in Line 3 be filled?

1) `Next = Parts.DeqRear()`  
2) `Out = Parts.DeqRear()`  
3) `Next = Parts.EnqRear()`  
4) `Out = Parts.DeqRear()`  
5) None of these

20. What is the order of the above function?

1) O(1)  
2) O(log n)  
3) O(n)  
4) O(n log n)  
5) O(n^2)  
6) O(n^2 log n)  
7) O(2^n)  
8) None of these

**intComma(-45002130) will print:**  
-45,002,130