CSC 212 Practice Midterm Exam 2B
Problems marked with (*) are challenging and problems marked with (**) are hard

Your Name: _____

1. (10 points) Consider the following linked list declaration. Assume pointers are 4 bytes, integers are 4 bytes and characters are 1 byte. Give a formula for the number of bytes used by a linked list of size n.

```
class LinkedList {
    struct Node {
        Node* next;
        char data[20];
        // ...
    }

    // ...

    Node* m_head;
    Node* m_tail;
    size_t m_size;

    // ...
}
```

Solution: The `LinkedList` class has three member variables, `m_head`, `m_tail` and `m_size`, each requiring 4 bytes for a total of 12 bytes. Each node uses 4 bytes for `next` and 20 bytes for `data`, for a total of 24 bytes. There are n nodes, so the total number of bytes is $24n + 12$.

2. (10 points) Implement the `max_multiplicity` functions. Assume `m_head` is the head of a **sorted** linked list containing zero or more elements. The **multiplicity** of an element is the number of times it occurs. Return the **maximum** multiplicity. Your implementation must run in $\mathcal{O}(n)$ time.

```
class Multiset {
    struct Node {
        Node* next;
        int data;
        // ...
    };

    // ...

    Node* m_head;

    // ...

    static size_t max_multiplicity(const Node* head) {
        // TODO: Implement this function.
    }

public:
    size_t max_multiplicity() const {
        // TODO: Implement this function.
    }
};
```

Solution:

```
class Multiset {
    struct Node {
        Node* next;
        int data;
        // ...
    };

    // ...

    Node* m_head;

    // ...

    // `max_multiplicity(head)` is the maximum multiplicity in the
    // sublist headed by `head`.
    static size_t max_multiplicity(const Node* head) {
        if (head == nullptr) return 0;
        size_t multiplicity = 1;
        int data = head->data;
        while (head->next && head->next->data == data) {
            head = head->next;
```

```
            ++multiplicity;
        }
        return std::max(multiplicity, max_multiplicity(head->next));
    }

public:
    // `max_multiplicity()` is the maximum multiplicity.
    size_t max_multiplicity() const {
        return max_multiplicity(m_head);
    }
};
```

3. (10 points) (*) Implement the `first_one` function **recursively**. Assume `v` contains zero or more 0's followed by one or more 1's. Return the index of the first 1. Your implementation must run in $\mathcal{O}(\lg n)$ time.

```
size_t first_one(const std::vector<int>& v) {
    // TODO: Implement this function.
}
```

Solution:

```
// `first_one(v, left, right)` is the index of the first `1` in
// `v[left:right]` where `v[left:right]` contains zero or more `0`s
// followed by one or more `1`s.
size_t first_one(const std::vector<int>& v, size_t left, size_t right) {
    if (right == left + 1) return left;
    size_t mid = left + (right - left - 1) / 2;
    if (v[mid] == 0) {
        return first_one(v, mid + 1, right);
    } else {
        return first_one(v, left, mid);
    }
}

// `first_one(v, left, right)` is the index of the first `1` in `v` where
// `v` contains zero or more `0`s followed by one or more `1`s.
size_t first_one(const std::vector<int>& v) {
    return first_one(v, 0, v.size());
}
```

4. (10 points) What does `L(6)` return?

```
int L(int n) {
    if (n == 0) return 2;
    if (n == 1) return 1;
    return L(n - 1) + L(n - 2);
}
```

Solution: $L(6) = 18$.

5. (10 points) Give a recurrence relation and base case for $T(n)$, the time complexity of the `mergesort` implementation provided below. Assume `merge` runs in $\Theta(n)$ time.

```
void merge(std::vector<int>& v, size_t left, size_t mid, size_t right) {
    // ...
}

void mergesort(std::vector<int>& v, size_t left, size_t right) {
    if (right <= left + 1) return;
    size_t mid = left + 2 * (right - left) / 3;
    mergesort(v, left, mid);
    mergesort(v, mid, right);
    merge(v, left, mid, right);
}

void mergesort(std::vector<int>& v) {
    return mergesort(v, 0, v.size());
}
```

Solution: `mergesort` makes two recursive calls, one of size $n/3$ and one of size $2n/3$, and performs $\Theta(n)$ additional work during `merge`. The base case occurs when $n = 0$ or $n = 1$ and performs a constant amount of work. Therefore, $T(n) = T(n/3) + T(2n/3) + \Theta(n)$ where $n \geq 2$, $T(1) = \Theta(1)$ and $T(0) = \Theta(1)$.

6. (10 points) Give a formula $C(n)$ for the number of complete 3-ary trees on n nodes.

Solution: $C(n) = 1$.

7. (10 points) (*) Find a closed form for $T(n) = 3T(n/2) + 1$ where $n \geq 2$ and $T(1) = 1$. Assume n is a power of 2.

Solution: The first expansion is

$$T(n) = 3T(n/2) + 1$$

The second expansion is

$$T(n) = 3^2 T(n/2^2) + 1 + 3$$

The third expansion is

$$T(n) = 3^3 T(n/2^3) + 1 + 3 + 3^2$$

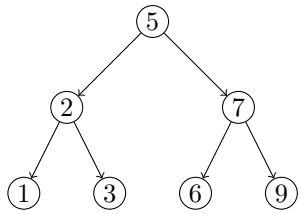
So the k-th expansion is

$$T(n) = 3^k T(n/2^k) + \sum_{i=0}^{k-1} 3^i$$

To reach the base case, $n/2^k = 1$, so $k = \lg n$. Then

$$\begin{aligned} T(n) &= 3^{\lg n} T(1) + \sum_{i=0}^{\lg n - 1} 3^i \\ &= 3^{\lg n} + \frac{3^{\lg n} - 1}{3 - 1} \\ &= \frac{3}{2} 3^{\lg n} - \frac{1}{2} \\ &= \frac{3}{2} 3^{\log_3 n / \log_3 2} - \frac{1}{2} \\ &= \frac{3}{2} n^{1/\log_3 2} - \frac{1}{2} \end{aligned}$$

8. (10 points) Delete $5, 7, 1$ from the following binary search tree. Use the **successor** replacement strategy. Draw the resulting tree after each deletion.



Solution: Check with `https://www.cs.usfca.edu/~galles/visualization/BST.html`.

9. (10 points) Insert $4, 8, 1, 6, 0, 3, 9$ into an initially empty 2-3-4 tree. Draw the resulting tree after each insertion.

Solution: Check with `https://www.cs.usfca.edu/~galles/visualization/BTree.html`.

10. (10 points) Explain the correspondence between red-black trees and 2-3-4 trees. Draw a diagram to illustrate your explanation.

Solution: Black nodes with no red children correspond to 2-nodes. Black nodes with one red child correspond to 3-nodes. Black nodes with two red children correspond to 4-nodes. Your answer should also include a drawing of a red-black tree and a equivalent 2-3-4 tree.