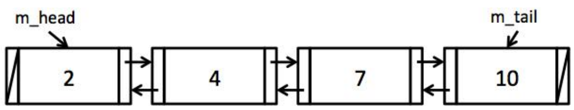
Name: **ID:**

Cautionary Rant: It is a good idea to check your work by actually programming any problem. But, it is critically important to complete these, or any exercise, without the aid of your computer **first**. You will not have access to a computer during the exam or during most technical interviews with any large software development company.

Problem 1: Doubly Linked List Class

We will build a sorted doubly linked list. The following shows an example of a list with 4 elements, where the nodes are sorted in the increasing order of their values. The `m_prev` pointer of the head node and `m_next` pointer of the tail node both point to `nullptr`. If the list is empty, head and tail pointers both point to `nullptr`.



Assume the following declaration of `Node` structure and `SortedLinkedList` class.

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

```
class SortedLinkedList {
public:
	SortedLinkedList();
	bool insert(const int &value);
	const Node *search(const int &value) const;
	void remove(Node *node);
	int size() const { return m_size; }
	void printIncreasingOrder() const;

private:
	Node * m_head;
	Node *m_tail;
	int m_size;
}
```

a. Implement SortedLinkedList().

```
SortedLinkedList::SortedLinkedList() {



}
```

Name: **ID:**

b. Implement `insert()`. If a node with the same value is already in the list, do not insert a new node. Return true if a new node is successfully inserted, and return false otherwise.

```
bool SortedLinkedList::insert(const int &value) {

}
```

c. Implement `search()`, which returns the pointer to the node with the specified value.

```
const Node *SortedLinkedList::search(const int &value) const {

}
```

Name: **ID:**

d. Implement `remove()`. Assume `node` is either `nullptr` (in which case you would simply return) or a valid pointer to a Node in the list, as found in `search()`.

```
void SortedLinkedList::remove(Node *node) {

}
```

e. Implement `printIncreasingOrder()`, which prints the values stored in the list in the increasing order, one value in each line.

```
void SortedLinkedList::printIncreasingOrder() const {

}
```

f. Program your implementation in a single file named sdll.h and include that with your quiz 2 submission archive.

Name: **ID:**

Problem 2: Recursion

a. Write a recursive function that computes base^{exp} (base raised to the power of exp). What is the Big-O of this function?

```
int recursivePow(int base, int exp) {

}
```

b. Write a function `powerThree` that, given a non-negative number `n`, returns 3^n (3^n, or "3 raised to power n") recursively, assuming 3^n is something that can be represented as an integer. Do not use a loop, and do not use the character `'*'` anywhere in your code. What is the Big-O of this function?

```
int powerThree(int n) {

}
```

Name: **ID:**

Problem 3: Big-O

a. Suppose Algorithm A and B perform the same task. For any given input size n, algorithm A executes in f(n)=0.003n2 operations, and algorithm B executes f(n)=250n operations. Specifically, when does Algorithm A perform better than B?

b. An algorithm that is $O(n^2)$ takes 10 seconds to complete when n=100. How long would you expect it to take when n=500?

c. What is the Big-O of the following code segment:

```
for (int i = 0; i < n; i++)
    for (int j = 0; j < 4 * i; j++)
        sum++;
```

d. What is the Big-O of the following function?

```
int gobidygoop(int n, int p) {
    int ac = 1;
    for (int i = 0; i < n; i++) {
        int k = p;
        while (k > 1) {
            ac *= i + k;
            k /= 4;
        }
    }
}
```

Name: **ID:**

e. Consider the following pseudo code:

```
Get value for n
Set the value of k to 1
While k is less than or equal to n
    Set the value of j to twice of k
    While j is greater or equal to 1
        Print the value of j
        Set the value of j to one half its former value;
    Increase k by 1
```

What is the Big-O of this pseudocode? What does this print if n is 4?

Problem 5: Sorting

a. Fill out the following table or sorting properties, if there is no special condition for a particular case then leave it blank:

Sorting Algorithm	Selection	Insertion	Bubble	Quick	Merge
Average Complexity					
Worse Complexity					
Condition for Worse					
Best Complexity					
Condition for Best					

Name: **ID:**

b. Sort the following array using the Mergesort algorithm. Show each recursive step, including the merge.

99	16	3	19	13	0	13	12