1. (5 marks) Dominant terms. What is dominant term in the following expressions? Logarithms are base 2.
   a. $0.01n^3 + 2n^5 + 100n$
   b. $3^n + 10n^4 + \log n$
   c. $0.001\log(\log n) + \frac{5}{n} + \log n$
   d. $n! + n^{10}$

2. (5 marks) Arrays and pointers. What are possible outputs of the following code? (Circle all correct answers)

   ```c
   #include <stdio.h>
   int main() {
       int m[2] = {5, 8};
       int * n = m;
       printf("m = %d, n = %d\n", m, n);
   }
   ```

   a) Program will not compile
   b) $m = 5, 8$, $n = 14285714$
   c) $m = 14285714, n = 14285714$
   d) $m = 5, n = 5$
   e) $m = 5, 8$, $n = 5, 8$

   14285714 is a possible address that $m$ points to, and $n$ points to the same location

3. (5 marks) Character arrays and strings. What are possible outputs of the following code? (Circle all correct answers)

   ```c
   #include <stdio.h>
   int main() {
       char x[5] = "Hello world!";
       printf("%s, ", x);
       printf("%c\n", x+7);
   }
   ```

   a) Program will not compile
   b) Hello world!, w
   c) Hello, w
   d) Hello, o

   Note: On Mac, b) is also possible, with all characters after “Hello” being random characters
4. (20 marks) Recursion. The following unfinished recursive implementation of binary search is supposed to return 1 if target is found in the array arr, and 0 otherwise. Complete the implementation by filling in the base case. Use comments to explain your code.

```c
int BinarySearch(int arr[], int len, int target) {
    if (len<=4) {
        for (int i = 0; i < len; i++) {
            if (arr[i] == target) {
                return 1;
            }
        }
        return 0;
    }
    int mid = len/2;
    return BinarySearch(arr,mid,target)+BinarySearch(arr+mid,len-mid,target);
}
```
5. (30 marks) Proof of correctness. Consider the following code and associated output.

<table>
<thead>
<tr>
<th>Code:</th>
<th>Output:</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>#include &lt;stdio.h&gt;</code></td>
<td>1</td>
</tr>
<tr>
<td><code>void fun(int N) {</code></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td><code>int z = 0;</code></td>
</tr>
<tr>
<td></td>
<td><code>for (int i = 0; i &lt; N; i++) {</code></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>}`</td>
</tr>
<tr>
<td><code>int main() {</code></td>
<td>64</td>
</tr>
<tr>
<td></td>
<td><code>fun(10);</code></td>
</tr>
<tr>
<td><code>}</code></td>
<td>100</td>
</tr>
</tbody>
</table>

a) What does the function output, for any given input integer N? (5 marks)

**The function prints all squares of 1 up to N.**

b) Prove your answer in part a) by applying the initialization, maintenance, and termination steps for proving program correctness. (20 marks)

*All the printing is done in the for loop, so it suffices to consider what happens in each iteration.*

*Initialization:* In the first iteration when `i = 0`, `z` starts at 0 and becomes `0 + (0 \times 2 + 1)`, which is 1. This is the correct value for the square of 1.

*Maintenance:* Suppose the last printed value of `z`, when `i = k - 1`, is `k^2`. In the next iteration, `i` would have a value of `k`, so `z` becomes `k^2 + 2k + 1`, which is `(k + 1)^2`. Therefore assuming the previous printed value is `k^2`, the next printed value would be `(k + 1)^2`.

*Termination:* The last value of `i` inside the for loop is `N - 1`. Since the assumption that `k^2` is printed leads to `(k + 1)^2` being printed is valid as proven in the maintenance step, and 1^2 is printed first as proven in the initialization step, we can conclude that in the last iteration, `N^2` will be printed. Therefore overall, the function prints `k^2` for `k` from 1 to `N` (inclusive).

c) What is the best big O estimate for the running time, with respect to the size of the input N? Give a brief justification to your answer. (5 marks)

*The loop is executed N times, and in each iteration, a fixed number of operations is performed. Therefore the function runs in O(N).*
6. Linked list. An implementation of the linked list is given below.

```c
#include <stdio.h>
#include <stdlib.h>

typedef struct _node {
    int data;
    struct _node * next;
} node_t;

typedef struct {
    node_t * head;
    node_t * tail;
} LL_t;

LL_t * LLcreate() {
    LL_t * ret = malloc(sizeof(LL_t));
    ret->head = NULL;
    ret->tail = NULL;
    return ret;
}

void LLappend(LL_t * intlist, int value) {
    node_t * newNode = malloc(sizeof(node_t));
    newNode->data = value;
    newNode->next = NULL;
    if (intlist->head == NULL) {
        intlist->head = newNode;
        intlist->tail = newNode;
    } else {
        intlist->tail->next = newNode;
        intlist->tail = newNode;
    }
}
```

Currently, two functions have been implemented: LLcreate and LLappend. In this question, you will implement two additional functions. Please use comments to explain your code.
a) (20 marks) Write a function called LLcomp that compares two linked lists. The function should return 1 if the two linked lists are equal in length and contains all the same elements in the same order. Otherwise, the function should return 0. The function prototype is given below:

```c
int LLcomp(LL_t * intlist1, LL_t * intlist2) {
    // Strategy: start at the end of the two lists, and go through elements one by one. At any point, if a discrepancy between the lists is found, return 0.
    node_t * curr1 = intlist1->head;
    node_t * curr2 = intlist2->head;

    // Go through the first list and compare with second list
    while (curr1 != NULL) {
        // If we arrive at the end of intlist2, but not at the end of intlist 1, then the two lists are not equal length
        if (curr2 == NULL) {
            return 0;
        }

        // If any element is not equal, the lists are not equal
        if (curr1->data != curr2->data) {
            return 0;
        }

        // Increment the pointers that go through the lists
        curr1 = curr1->next;
        curr2 = curr2->next;
    }

    // When we finish going through intlist1, if intlist2 still has more elements, then the two lists are not equal
    if (curr2 != NULL) {
        return 0;
    }

    // If we made it all the way here, the two lists are equal
    return 1;
}
```
b) (15 marks) Write a function called LLupdate that updates the value of a node in a linked list. If the node is not found, the linked list should not change; instead, the user should get the following message: “Node not found!” The function prototype is given below:

```c
void LLupdate(LL_t * intlist, node_t * n, int value) {
    // We go through each node in the list, and if we find a node that is equal to a node in intlist, we update its value
    // This time we will do a for loop to illustrate its use
    for (node_t * curr = intlist->head; curr != NULL; curr = curr->next) {
        if (curr == n) {
            curr->data = value;
        }
    }
}
```