1. (10 marks) Dominant terms. What is dominant term in the following expressions? Logarithms are base 2.
   a. \(0.01n^{10000} + 1.001^n\)
   b. \(5n^{1001} + 2\log n\)
   c. \(10\log(\log n) + \log(10n)\)
   d. \(n! + 2^n + n^{10000}\)

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[7] = {1, 1, 5, 5, 6, 6, 5};
       printf("%d, %d, %d, %d\n", sizeof(int), m, m[4], &m[4]);
   }
   ```

   a) 4, 571428, 571432, 6
   b) 4, 235711, 6, 235727
   c) 4, 1, 6, 235711
   d) 4, 112358, 6, 112362

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 greet[14] = "Hello world!";
       for (int i = 0; i < 13; i++) {
           printf("%d ", greet[i]);
           i++;
       }
       printf("\n");
   }
   ```

   a) 72 108 111 119 114 100 0
   b) Hello world!
   c) Hello world!!!
   d) 72 108 111 119 114 100
void QuickSort(int arr[], int first, int last) {
    // Base case
    if (last <= first + 1) {
        // Here, the array may contain 0, 1, or 2 elements
        if (first == last - 1 && arr[first] > arr[last]) {
            // If array contains 2 elements, swap if they’re in the wrong order
            int tmp = arr[first];
            arr[first] = arr[last];
            arr[last] = tmp;
        }
        // If array contains 0 or 1 elements, don’t do anything
        return;
    }
    // Split array
    int mid = partition(arr, first, last);
    // Recurse
    QuickSort(arr, first, mid-1);
    QuickSort(arr, mid+1, last);
}
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>88</td>
</tr>
<tr>
<td><code>void very_fun(int arr[], int N) {</code></td>
<td>9</td>
</tr>
<tr>
<td><code>for (int i = 0; i &lt; N-1; i++) {</code></td>
<td>7</td>
</tr>
<tr>
<td><code>    for (int j = N-1; j &gt; i; j--) {</code></td>
<td>6</td>
</tr>
<tr>
<td><code>        if (arr[j-1] &lt; arr[j]) {</code></td>
<td>5</td>
</tr>
<tr>
<td><code>            int tmp = arr[j];</code></td>
<td>4</td>
</tr>
<tr>
<td><code>            arr[j] = arr[j-1];</code></td>
<td>3</td>
</tr>
<tr>
<td><code>            arr[j-1] = tmp;</code></td>
<td>2</td>
</tr>
<tr>
<td><code>        }</code></td>
<td>1</td>
</tr>
<tr>
<td><code>}</code></td>
<td></td>
</tr>
<tr>
<td><code>int main() {</code></td>
<td></td>
</tr>
<tr>
<td><code>    int arr[9] = {4, 5, 7, 1, 3, 88, 6, 2, 9};</code></td>
<td></td>
</tr>
<tr>
<td><code>    very_fun(arr, 9);</code></td>
<td></td>
</tr>
<tr>
<td><code>    for (int i = 0; i &lt; 9; i++) {</code></td>
<td></td>
</tr>
<tr>
<td><code>        printf(&quot;%d\n&quot;, arr[i]);</code></td>
<td></td>
</tr>
<tr>
<td><code>}</code></td>
<td></td>
</tr>
</tbody>
</table>

a) What does the function `very_fun` output, for any given input array and its length? (5 marks)

This function sorts an array in descending order.

b) Prove your answer in part a) by applying the initialization, maintenance, and termination steps for proving program correctness. Apply your argument to the outer loop involving i. (20 marks)

The inner loop starts at the end of the array, and swaps element j with element j-1 if element j is smaller. This continues until elements i+1 and i are compared and possibly swapped. Thus, one can expect the invariant in the outer loop to be that elements (at index) 0 to i-1 are sorted in descending order, and larger than element i and all elements after i. We will now prove this.

**Initialization:** When i = 0, the invariant is trivially true.

**Maintenance:** Suppose that in the beginning of the outer loop when i = k for some non-negative k, we have that arr[0...k-1] are sorted in descending order, and all elements from arr[0...k-1] are larger than elements in arr[k...N-1]. In the inner loop, larger elements are swapped towards the left (smaller indices) until the largest element in arr[k...N-1] is placed at arr[k]. Therefore, after the outer loop iteration with i=k, elements in arr[0...k] would be larger than those in arr[k+1...N-1]. In addition, since arr[k] is smaller than elements in arr[0...k-1] by assumption, arr[0...k] must be sorted in descending order.
**Termination:** After the outer loop finishes, we have \( i = N - 1 \), so \( \text{arr}[0...N-2] \) will be sorted in descending order, and all its elements are greater than \( \text{arr}[N-1] \). Thus we can conclude that the entire array is sorted in descending order.

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)

In the outer loop, \( i \) goes from 0 to \( N-2 \). In the inner loop, \( j \) goes from \( N-1 \) to \( i+1 \), which is \( N-1-(i+1)+1 = N-i-1 \) iterations. Thus, the total number of iterations is \( \sum_{i=0}^{N-2} (N - i - 1) = N - 1 + N - 2 + \cdots + 2 + 1 \). This is \( O(N^2) \), since the number of iterations in the inner loop decreases linearly with \( i \), with a maximum of \( N - 1 \approx N \).

Mathematically, we have \( N - 2 + N - 3 + \cdots + 1 + 0 = (0 + N) \left( \frac{N-2+1}{2} \right) = O(N^2) \) (where we did not need to simplify fully to see the \( N^2 \)).
6. Linked list. An implementation of the linked list with only a pointer to the first node 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;
} LL_t;

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

Currently, only one function has been implemented: LLcreate. In this question, you will implement two additional functions without changing the code above. Please use comments to explain your code.
void LLappend(LL_t * intlist, int value) {
    // Create the new node
    node_t * newNode = malloc(sizeof(node_t));
    newNode->data = value;
    newNode->next = NULL;

    // If the list is empty, update the head pointer
    if (intlist->head == NULL) {
        intlist->head = newNode;
        return;
    }

    // If the list is nonempty, traverse along the list to the end
    // while keeping track of the previous node
    node_t * prev = intlist->head;
    node_t * curr = intlist->head->next;

    while (curr != NULL) {
        curr = curr->next;
        prev = prev->next;
    }

    // Once the "curr" pointer finishes traversing the array,
    // the "prev" pointer would point to the last node, whose next
    // pointer should be updated
    prev->next = newNode;
}
b) (10 marks) Write a function called LLlength that returns the number of elements in the list.
The function prototype is given below.

```c
int LLlength(LL_t * intlist) {
    // Strategy: use the variable length to keep track of number of
    // elements, and use a for loop to go through the list while
    // incrementing length, until we reach NULL
    int length = 0;
    for (node_t * curr = intlist->head;
        curr != NULL;
        curr = curr->next) {
        length++;
    }

    return length;
}
```