Ryan Christopher
MET CS566
Assignment 2

1. For each of the following pairs of functions, either f(n) is in O(g(n)), f(n) is in Ω(g(n)), or f(n) = Θ(g(n)). Determine which relationship is correct and briefly explain why.

a) f(n) = log(n^2) g(n) = log(n) + 5

f(n) = Ω(g(n))
After a certain n_0 and constant c, f(n) will always lie on or above g(n).

b) f(n) = $\sqrt{n}$ g(n) = log(n^2)

f(n) = Ω(g(n))
After a certain n_0 and constant c, f(n) will always lie on or above g(n).

c) f(n) = n g(n) = $n$log(2n)

f(n) = O(g(n))
After n_0 = 5 and c = 1, f(n) will always lie on or below g(n).

d) f(n) = 10 g(n) = log(10)

f(n) = Ω(g(n))
Both f(n) and g(n) are constants, and f(n) will always lie above g(n).

e) f(n) = 2^n g(n) = 10n^2

f(n) = O(g(n))
At n_0 = 10 and constant c = 1, f(n) will always lie on or below g(n).

2. Use Master Theorem, Master Theorem Extension or Substitution Method

a) $T(n) = 2T(n / 3) + n^3$
a = 2 b = 3 d = 3
$d > \log_3(2)$
Case 3: $T(n) = O(n^3)$

b) $T(n) = T((9/10)n) + \sqrt{n}$
a = 1 b = 10/9 d = ½
$d > \log_{10/9}(1)$
Case 3: $T(n) = O(n^{1/2})$

c) $T(n) = 16T(n/4) + n^2$
a = 16 b = 4 d = 2
$d = \log_4(16)$
Case 2: $T(n) = O(n^2 \log(n))$

d) $T(n) = T(n - 3) + n$
a = 1 b = 3 d = 1
a = 1
Case 2: $T(n) = O(n^{d+1}) = O(n^2)$

e) $T(n) = 3T(n - 1) + 1, T(1) = 1$
a = 3 b = 1 d = 0
a > 1
Case 1: $T(n) = O(n^0 3^{n/1}) = O(3^n)$

3. How many inversions (swaps) requires sorting the following lists? A pair of elements (p_i, p_j) is called an inversion in a list permutation p if $i > j$ and $p_i < p_j$. Show your work in details, name a sorting algorithm in use.

The sorting algorithm in use for the following lists is Bubble Sort, as the way the algorithm works is inverting a value with the adjacent value as long as they are in an incorrect ordering, then starting back at the beginning. This process will repeat until a full iteration of the list is performed and there are no inversions.

a) (5, 2, 3, 4, 1)

Total inversions required: 7

List	p_i	p_j	i	j	$i > j$ and $p_i < p_j$
5, 2, 3, 4, 1	2	5	1	0	True
2, 5, 3, 4, 1	3	5	2	1	True
2, 3, 5, 4, 1	4	5	3	2	True
2, 3, 4, 5, 1	1	5	4	3	True
2, 3, 4, 1, 5	3	2	1	0	False
2, 3, 4, 1, 5	4	3	2	1	False
2, 3, 4, 1, 5	1	4	3	2	True
2, 3, 1, 4, 5	5	4	4	3	False
2, 3, 1, 4, 5	3	2	1	0	False
2, 3, 1, 4, 5	1	3	2	1	True
2, 1, 3, 4, 5	4	3	3	2	False
2, 1, 3, 4, 5	1	2	1	0	True
1, 2, 3, 4, 5	2	1	1	0	False
1, 2, 3, 4, 5	3	2	2	1	False
1, 2, 3, 4, 5	4	3	3	2	False
1, 2, 3, 4, 5	5	4	4	3	False

b) (1, 2, 3, 4, 5)

Total inversions required: 0

List	p_i	p_j	i	j	$i > j$ and $p_i < p_j$
1, 2, 3, 4, 5	2	1	1	0	False
1, 2, 3, 4, 5	3	2	2	1	False
1, 2, 3, 4, 5	4	3	3	2	False
1, 2, 3, 4, 5	5	4	4	3	False

c) (5, 1, 2, 3, 4)

Total inversions required: 4

List	p_i	p_j	i	j	$i > j$ and $p_i < p_j$
5, 1, 2, 3, 4	1	5	1	0	True
1, 5, 2, 3, 4	2	5	2	1	True
1, 2, 5, 3, 4	3	5	3	2	True
1, 2, 3, 5, 4	4	5	4	3	True
1, 2, 3, 4, 5	2	1	1	0	False
1, 2, 3, 4, 5	3	2	2	1	False
1, 2, 3, 4, 5	4	3	3	2	False
1, 2, 3, 4, 5	5	4	4	3	False

4. Let us consider one of the easiest sorting algorithms – Maxsort. It works as follows: Find the largest key, say max, in the unsorted section of array (initially the whole array) and then interchange max with the element in the last position in the unsorted section. Now max is considered part of the sorted section consisting of larger keys at the end of the array. It is no longer in the unsorted section. Repeat this until the whole array is sorted.

a) Write an algorithm for Maxsort assuming an array E contains *n* elements to be sorted, with indexes 0, 1, ... , *n* – 1.

The algorithm for this problem is located in the file main.rs submitted along with this paper. Below is the function max_sort containing the algorithm:

```rust
// Problem 4:
// max_sort takes as input an array of integers and returns the array sorted from smallest to largest
fn max_sort(arr: &mut [i32]) {
    println!("Starting:\n{:?}", arr);
    // store number of comparisons
    let mut comps = 0;
    // set the end value to the last entry in the array
    let mut end_val = arr.len() - 1;
    // while the last value is not the beginning, repeat
    while end_val >= 1 {
        // set curent max and index to first value of array
        let mut max = arr[0];
        let mut index = 0;
        // iterate until the end value of the unsorted vals
        for num in 0..=end_val {
            // find largest value and store corresponding index
            comps += 1;
            if arr[num] > max {
                max = arr[num];
                index = num;
            }
        }
        // create temp variable to store the last unsorted val, then replace with
        // then swap the last val with the max found
        let temp = arr[end_val];
        arr[end_val] = max;
        arr[index] = temp;
        // decrement end_val by one as there is one more value sorted, continue until beginning of array
        end_val -= 1;
        println!("{:?}", arr);
    }
    println!("Sorted:\n{:?}", arr);
    println!("Comparisons: {}", comps);
}
```

b) How many comparisons of keys does Maxsort do in the worst case and on average?

Due to how this algorithm processes the array, the worst case and average case are the same number of comparisons and can be calculated by $\sum_{i=0}^{n} i + 1$. This is due to the algorithm adding 1 element to the “sorted” list with each iteration. By this logic, there are n comparisons to find the max on the first iteration, n – 1 comparisons on the second, n – 2 on the third, and so on until there is only 1 value left which will yield a total number of comparisons equal to the sum of the length of array + 1.

In the test cases submitted in the source code, there is a sorted array, unsorted array, and random array. Regardless of their initial order, the algorithm will always perform a number of comparisons equal to the sum of the length of the array + 1. Below is the output:

```
Starting:
[1, 2, 3, 4, 5]
[1, 2, 3, 4, 5]
[1, 2, 3, 4, 5]
[1, 2, 3, 4, 5]
[1, 2, 3, 4, 5]
Sorted:
[1, 2, 3, 4, 5]
Comparisons: 14
Starting:
[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
[1, 9, 8, 7, 6, 5, 4, 3, 2, 10]
[1, 2, 8, 7, 6, 5, 4, 3, 9, 10]
[1, 2, 3, 7, 6, 5, 4, 8, 9, 10]
[1, 2, 3, 4, 6, 5, 7, 8, 9, 10]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Sorted:
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Comparisons: 54
Starting:
[12, 2, 58, 9, 100, 57, 79]
[12, 2, 58, 9, 79, 57, 100]
[12, 2, 58, 9, 57, 79, 100]
[12, 2, 57, 9, 58, 79, 100]
[12, 2, 9, 57, 58, 79, 100]
[9, 2, 12, 57, 58, 79, 100]
[2, 9, 12, 57, 58, 79, 100]
Sorted:
[2, 9, 12, 57, 58, 79, 100]
Comparisons: 27
```

5. Come up with an algorithm to convert 2025 using Roman numerals. Write a code for it. What about 3998 and 5999? Please provide Roman representation of all input numbers.

The code for this has been submitted in the file main.rs along with this paper.

```rust
// Problem 5:
// roman_numeral takes as input a decimal value and ouptuts the corresponding Roman numeral
// representation of the input
fn roman_numeral(mut value: i32) -> String{
    // set roman numeral to empty string
    let mut roman_num: String = "".to_owned();
    // create arrays for representation values and representations
    let rep_vals: [i32; 13] = [1000, 900, 500, 400, 100, 90, 50, 40, 10, 9, 5, 4, 1];
    let rep: [&str; 13] = ["M", "CM", "D", "CD", "C", "XC", "L", "XL", "X", "IX", "V", "IV", "I"];
    // until value is 0, find largest match in rep_vals and add the corresponding rep to
    // roman_num to be returned
    while value > 0 {
        for num in 0..=rep_vals.len() - 1 {
            if value ≥ rep_vals[num] {
                roman_num.push_str(rep[num]);
                value -= rep_vals[num];
                break;
            }
        }
    }
    // return roman numeral representation
    roman_num.to_string()
}
```

When provided the input numbers of 2025, 3998, and 5999, the algorithm produced the following outputs:

```
Decimal: 2025
Roman  : MMXXV
Decimal: 3998
Roman  : MMMCMXCVIII
Decimal: 5999
Roman  : MMMMMCMXCIX
```