CS 2210 Data Structures and Algorithms
Assignment 1
Ian Borwick 250950449

1. To show that $3n^3$ is $O(n^4)$ we must find constants $c > 0$ and $n_0 \geq 1$ such that:

$$3n^3 \leq cn^4, \qquad \forall n \geq n_0$$

First, we simplify the inequality by moving the n^3 term to the right-hand side by dividing both sides (we know we can do this because $n > n_0 > 0$) in order to the following:

$$3 \leq cn, \qquad \forall n \geq n_0$$

Since the right-hand side of the inequality need to be positive, we choose $c = 3$:

$$3 \leq 3n = n, \qquad \forall n \geq n_0$$

Note that $n \geq 1$ for all values $n \geq 1$, so we choose $n_0 = 1$,
As we have found constant value $c = 3$ and $n_0 = 1$ that make inequality (1) true, then we have proven that $3n^3$ is $O(n^4)$.

2. To show that n^3+n^2 is not $O(n^2)$ we use a proof through contradiction. Assume that n is $O(n^2)$ and derive a contradiction using this assumption. If n^3+n^2 is $O(n^2)$ that means that we need to find constants $c > 0$ and $n_0 \geq 1$ such that:

$$n^3+n^2 \leq cn^2, \qquad \forall n \geq n_0$$

Dividing the entire inequality by n^2 (which can be done since $n \geq n_0 \geq 1$) we get the following:

$$n+1 \leq c, \qquad \forall n \geq n_0$$

This cannot hold true as n will grow without bound as c stays constant. Therefore, there is no value of c or n_0 that will make this inequality true as the upper bound of c will be far surpassed very quickly. As such we can determine that n^3+n^2 is not $O(n^2)$.

3. Given f(n) and g(n) are non-negative integer, and that f(n) is O(g(n)) and g(n) is O(f(n)). We can prove that f(n)-g(n) is O(f(n)).

As f(n) is O(g(n) there exists a $c > 0$ and $n_0 \geq 1$ such that:

$$f(n) \leq c'g(n), \qquad \forall n \geq n_0$$

As g(n) is O(g(n) there exists a $c > 0$ and $n_0 \geq 1$ such that:

$$g(n) \leq c''f(n), \quad \forall n \geq n_0$$

We want $c'' > 0$ and $n_0'' \geq 1$ such that:

$$f(n) - g(n) \leq c''f(n), \quad \forall n \geq n_0$$

Let $n_0'' = \max(n_0, n_0')$ thus for all $n \geq n_0$ we have the following:

$$f(n) - g(n) \leq cg(n) - g(n) = (c-1)\, g(n) \leq (c-1)c'f(n), \quad \forall n \geq n_0$$

Take $c'' = (c-1)\, c'$, thus $f(n) - g(n)$ is $O(f(n))$

4. i)

Algorithm: CompareArray(A,B)
In: Array A storing n integer values and array B storing n integer values
Out: False if any value of A is the same as any value, and true if there are any common values

```
{
    boolean bool = true
    int i =0
    int j = 0
    while i < A.length do {
        while j < B.length do {
            if A[i] == B[j] do {
                bool = false
                break loop
            }
            j=j+1
        }
        If bool = false do {
            break loop
        }
        j=0
        i=i+1
    }
    return bool
}
```

ii)a. The outside loop is repeated for each value of i between 0 and the length of array A. Hence, the outside loop is repeated n times. Which would grow uncontrollably if it were not restricted by the statement i < A.length. The inside loop is repeated for each value of j between

0 and the length of array. Hence the inside loop is iterated n times. Which would grow uncontrollably if it were not restricted by the statement j < B.length. Therefore, the total amount of iterations is at most n^2. The algorithm then terminates after a finite amount of time as i=n.

b. The value of bool is returned and set to true to start. Then first loop considers $i \in \{0, 1, 2, \ldots\ldots, n-1\}$. For each of these values the second loop considers $j \in \{0,1,2,3,\ldots\ldots,n-1\}$. Within the inner loop the algorithm compares the values of i and j, through the if statement and if the values are the same the value of bool will be changed to false and the loop will break or, the interior loop is then iterated to the next value for comparison and so forth. Once the interior loop has checked all values at position j the value of i is incremented by 1 and the value of j = 0 and the value of bool is checked if it is false the loop will break exiting the iterations. Or this starts the second iteration of the outer loop comparing the value at the next position and resets the interior loop to be iterated again. It can be seen that every value of A[i] is compared to every value of B[j] thus the value of bool will only be changed if there are common values. The algorithm then returns the value of bool which will be true if there are no common values and false if there are thus the value returned is correct

iii) The worst-case situation for this algorithm is if the common values are in the n-1 position of both arrays or if there are no common values. This would force the algorithm to do the maximum amount of iterations before returning a value and as a result would give the largest time complexity.

iv) We analyze the inner-most while loop first. In every iteration this loop performs a constant number c_3 of operations and the loop is iterated n times. Thus, the total number of operations performed by the inner loop is c(n).

The outer-most while loop iterates n times with c_2 constant operations performed. And the remainder after the while loops performs a constant c_4 operations. In addition to the initial constant c_1. Due to this the algorithm in the worst case performs $c_1n(c_2n) + c_3 + c_1$. This is equal to $c_5n^2+c_6$. The total number of operations in the worst case is $c_5n^2+c_6$ is $O(n^2)$.

5. A) Factorial Search

Size of Input	Running times ns
7	3003616
8	19718237
9	76888064
10	842508870
11	8888677046
12	10283297440

b) Quadratic Search

Size of Input	Running times ns

5	426
10	1183
100	15030
1000	142321
2000	584000

c) Linear Search

Size of Input	Running times ns
5	194
10	382
100	1791
1000	5821
2000	9228
10000	12534