## CS 2210 Data Structures and Algorithms
## Solution for Assignment I

1. To show that $1/n^2$ is $O(1/n)$, we need to find constants $c > 0$ and $n_0 \geq 1$ such that

$$\frac{1}{n^2} \leq c\frac{1}{n}, \qquad \forall n \geq n_0.$$

(The symbol $\forall$ means "for all".) Since we only consider values $n \geq n_0 \geq 1$ then we can multiply both sides of the above inequality by $n^2$ to get

$$1 \leq cn \quad \forall n \geq n_0.$$

We can now choose, for example, $c = 1$ and the above inequality becomes

$$1 \leq n \quad \forall n \geq n_0.$$

The inequality is true for all values $n \geq 1$, so we can choose $n_0 = 1$.

2. To show that $1/g(n)$ is $O(1/f(n))$, we must find constant values $c > 0$ and $n_0 \geq 1$ such that

$$\frac{1}{g(n)} \leq c\frac{1}{f(n)}, \qquad \forall n \geq n_0 \tag{1}$$

To find these values for $c$ and $n_0$ we use the fact that $f(n)$ is $O(g(n))$, which means that there are constant values $c' > 0$ and $n_0' \geq 1$ such that

$$f(n) \leq c'g(n), \qquad \forall n \geq n_0', \tag{2}$$

Since $f(n)$ and $g(n)$ are non-negative functions, if we divide both sides of inequality (2) by $f(n) \times g(n)$ we get

$$\frac{1}{g(n)} \leq c'\frac{1}{f(n)} \qquad \forall n \geq n_0'. \tag{3}$$

Observe that inequality (3) has the same form as inequality (1), hence we can choose $c = c'$ and $n_0 = n_0'$ noting that $c'$ and $n_0'$ are constants. Since we have found constant values $c$ and $n_0$ which make inequality (1) true, then we have shown that $f(n)/g(n)$ is $O(1)$.

3. To show that $n-1$ is not $O(1)$ we use a proof by contradiction: We assume that $n-1$ is $O(1)$ and derive a contradiction from this assumption. If $n-1$ is $O(1)$ then there are constants $c > 0$ and $n_0 \geq 1$ for which

$$n - 1 \leq c, \qquad \forall n \geq n_0.$$

Add 1 to both sides of the inequality to get

$$n \leq c + 1, \qquad \text{for all } n \geq n_0 \tag{4}$$

Note that this last inequality cannot be true because $c + 1$ is a constant but $n$ grows without bound, hence it cannot be true that for all values of $n$ larger than any constant $n_0$ the value of $n$ is at most a constant $c$. Therefore, since inequality (4) is a contradiction we have shown that $n-1$ is not $O(1)$.

4. ```
public int most_times(int[] A, int n) {
// Input:  Array A storing n integer values
// Output:  The value that appears the largest number of times in
// A.  If several values appear in A the largest number of
// times, the algorithm must return the smallest among these values.
  int pos_max = -1; // Position of the value that appears the maximum number
                    // of times in A
 int max_count = 0; // Number of times that A[pos_max] appears in A
 int count = 0;
 for (int i = 0; i < n; ++i) {
    count = 0;
    for (int j = i; j < n; ++j) // Count the number of times that A[i] appears
                                 // in A[i..n-1]
        if (A[i] == A[j]) ++count;

    // Select the value that appears the maximum number of times, breaking
    // ties in favor of smaller values
    if ((count>max_count) || (count == max_count && A[i] < A[pos_max])) {
        max_count = count;
        pos_max = i;
    }
 }
 return A[pos_max];
}
```

5. (*Proof of termination*) The algorithm does not always terminate. Consider an array $L$ storing $n$ integer values and a value $x$ that is not in $L$. The condition of the first **if** statement (**if** $L[i] == x$) will never be true, so each iteration of the **while** loop will increase the value of $i$ by 2.

   Therefore, the variable $i$ will take values 0, 2, 4, ..., but when $i$ takes value $n-1$ (if $n$ is odd) or $n$ (if $n$ is even), the second **if** statement (**if** $i \geq n-1$) will set the value of the variable $i$ to 1, so in the following iterations of the loop the value of $i$ will be 1, 3, 5, ... This time when $i$ takes value $n-1$ (when $n$ is even), or $n$ (when $n$ is odd) the second **if** statement will again set the value of $i$ to 1 and hence the algorithm will never terminate.

6. (*Proof of correct output*) We need to prove 2 things:

   - If $x$ is not in $L$ the algorithm must return -1. Note that initially the variable $j$ has value 0 and in each iteration of the while loop the value of $j$ increases by 2 since the condition of the first **if** statement will always be false, given that $x$ is not in $L$. We consider two cases.
     - Consider first the case when $n$ is even. The variable $j$ will take values 0, 2, 4, ..., $n-2$, $n$; when $j = n$ the condition of the second **if** statement (**if** $j \geq n-1$) will
```

be true, so the value of $j$ will be set to 1. Note that so far the algorithm will have compared the value of $x$ with all values in $L$ stored in even positions of the array. In the following iterations of the **while** loop the variable $j$ takes values 1, 3, 5, ..., $n-1$ so the value of $x$ is compared with all values stored in odd positions of the array $L$. At this point the value of $x$ will have been compared with all values in $L$ and so the value of variable $i$ will be equal to $n$ and therefore the **while** loop terminates and the algorithm correctly returns the value -1.

  - The case when $n$ is odd is very similar. Variable $j$ takes values 0, 2, 4, ..., $n-1$, 1, 3, 5, ... $n-2$ and then the **while** loop terminates. Since $x$ is compared with all values in $L$, the algorithm will correctly return the value -1.

- If $x$ is in $L$ the algorithm must return the position of $x$ in $L$. By the above argument, the algorithm compares the value $x$ with all values stored in array $L$. Therefore if $x$ is in $L$ the condition of the first **if** statement will be true (**if** $L[j] = x$) and therefore the algorithm will correctly return the position $i$ of $x$ in $L$.

7. Here are tables giving possible running times for the algorithms.

| n | Linear Search |
|---|---|
| 5 | 27 ns |
| 10 | 55 ns |
| 100 | 496 ns |
| 1000 | 3498 ns |
| 100000 | 32843 ns |

| n | Quadratic Search |
|---|---|
| 5 | 196 ns |
| 10 | 774 ns |
| 100 | 6045 ns |
| 1000 | 168330 ns |
| 10000 | 16150843 ns |

| n | Factorial Search |
|---|---|
| 7 | 2159000 ns |
| 8 | 11615700 ns |
| 9 | 52594100 ns |
| 10 | 540661500 ns |
| 11 | 6828988100 ns |
| 12 | 84883873100 ns |