

**CS 2210 Data Structures and Algorithms**  
**Solution for Assignment 1**

1. To show that  $3n^2 + 4n$  is  $O(n^2)$  we must find constants  $c > 0$  and  $n_0 \geq 1$  such that

$$3n^2 + 4n \leq cn^2, \quad \forall n \geq n_0$$

Let us move  $3n^2$  to the right hand side to get

$$4n \leq (c - 3)n^2, \quad \forall n \geq n_0.$$

Now, we divide both sides of the first inequality by  $n$  (this can be done since  $n$  is positive as  $n \geq n_0 \geq 1$ ), to get

$$4 \leq (c - 3)n, \quad \forall n \geq n_0.$$

Finally, we can choose for  $c$  any value larger than 3, for example  $c = 4$ , and the above inequality becomes

$$4 \leq n, \quad \forall n \geq n_0$$

from which we see that we can select  $n_0 = 4$ , as the first inequality,  $4 \leq n$ , holds for all values  $n \geq 4$ .

2. To show that  $n$  is not  $O(\sqrt{n})$  we have to prove that it is **not** possible to find constant values  $c > 0$  and  $n_0 \geq 1$  such that

$$n \leq c \times \sqrt{n}, \quad \forall n \geq n_0.$$

This is equivalent to show that for **every** constant values  $c > 0$  and  $n_0 \geq 1$ ,

$$n > c \times \sqrt{n}, \quad \text{for at least one value } n \geq n_0. \tag{1}$$

Let us divide both sides of the above inequality by  $\sqrt{n}$  (again this can be done since  $n \geq n_0 \geq 1$ , so  $\sqrt{n}$  is positive), to get

$$\sqrt{n} > c, \quad \text{for at least one value } n \geq n_0.$$

Taking squares on both sides (this can be done as both sides are positive), we get

$$(\sqrt{n})^2 = n > c^2, \quad \text{for at least one value } n \geq n_0.$$

Finally, note that  $n > c^2$  for all values  $n \geq \max\{c^2, n_0\}$ .

**Alternative proof**

We can also use a proof by contradiction. Assume that  $n$  is  $O(\sqrt{n})$  and derive a contradiction from this assumption. If  $n$  is  $O(\sqrt{n})$ , this means that we can find constants  $c > 0$  and  $n_0 \geq 1$  for which

$$n \leq c \times \sqrt{n}, \quad \forall n \geq n_0.$$

Dividing both sides by  $\sqrt{n}$ , we get

$$\sqrt{n} \leq c, \quad \forall n \geq n_0,$$

which clearly cannot hold, since  $\sqrt{n}$  grows without bound, so  $\sqrt{n}$  cannot be upper-bounded by any constant  $c$ . Therefore,  $n$  is not  $O(\sqrt{n})$ .

3. To show that  $k \times f(n) - k' \times g(n)$  is  $O(f(n))$  we must find constant values  $c > 0$  and  $n_0 \geq 1$  such that

$$k \times f(n) - k' \times g(n) \leq cf(n), \quad \forall n \geq n_0 \tag{2}$$

Moving  $k \times f(n)$  to the right hand side of the inequality we get

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

Note that since  $k'$  is a positive constant and  $g(n)$  is a non-negative function, then  $-k' \times g(n) \leq 0$ . Hence, if we choose  $c > k$  we ensure that

$$(c - k)f(n) \geq -kg(n).$$

Selecting, for example,  $c = k + 1$  and  $n_0 = 1$  ensures that the inequality holds.

4 (i) **Algorithm**  $k$ -flat( $A, n, k$ )

**In:** Array  $A$  of size  $n$  and value  $k$

**Out:** True if  $A$  is  $k$ -flat; false otherwise

```

{
  for  $i \leftarrow 0$  to  $n - 1$  do {
    // Try to find a value  $A[j]$ ,  $j \neq i$  such that  $A[j] + A[i] = k$ .
    pairFound  $\leftarrow$  false
    for  $j \leftarrow 0$  to  $n - 1$  do
      if ( $i \neq j$ ) and ( $A[j] + A[i] = k$ ) then pairFound  $\leftarrow$  true

    if pairFound = false then return false
  }
  return true
}

```

4 (ii) The worst case for the algorithm is when the array is  $k$ -flat as in this case the condition of the second **if** statement is never true and so the **for** loops perform the maximum number of iterations.

We analyze the inner-most **for** loop first. In every iteration of this loop a constant number  $c$  of operations is performed and the loop is repeated  $n$  times. Thus, the total number of operations performed by this loop is  $cn$ .

As for the first **for** loop, in every iteration it performs a constant number of operations  $c'$  to update and test the value of  $i$ , to set **pairFound** to **false**, and to check at the end of each iteration whether **pairFound** = **false**, plus the  $cn$  operations of the second **for** loop. The first loop is repeated  $n$  times, thus the total number of operations performed in this loop is

$$n(c' + cn) = cn^2 + c'n.$$

There is a constant number of operations  $c''$  performed after the first **for** loop, so the total number of operations performed by the algorithm is  $cn^2 + c'n + c''$ , which is  $O(n^2)$ .

5. We analyze the inner-most **for** loop first. In each iteration of the loop only a constant number  $c$  of operations is performed. Hence, the total number of operations performed by this loop is  $c(i + 1)$ , as the loop is repeated for all values of  $j$  from 0 to  $i$ . This **for** loop is inside the second **for** loop, which in each iteration performs an additional constant number  $c'$  of operations to update the value of  $i$  and to decide whether the upper bound for  $i$  has been reached. Thus, the total number of operations performed by the outer **for** loop is

$$\sum_{i=0}^{n^2} (c' + c(i + 1)) = \sum_{i=0}^{n^2} (c + c') + c \sum_{i=0}^{n^2} i = (c + c')(n^2 + 1) + c \frac{n^2(n^2 + 1)}{2} = c + c' + (c + c' + \frac{c}{2})n^2 + \frac{c}{2}n^4$$

There is an additional constant number  $c''$  of operations performed outside the loops, so the total number of operations performed by the algorithm is  $c + c' + c'' + (c + c' + \frac{c}{2})n^2 + \frac{c}{2}n^4$ . To compute the order of this time complexity we discard constant terms and we keep the fastest growing term, to get that the time complexity of the algorithm is  $O(n^4)$ .