Exercise 2 – Solutions

1. Ex1: $O(n)$ (simple loop over all elements of a list)

   Ex2: $O(n)$ If every second element is skipped, it changes the constant, but not the asymptotic rate of growth, of the operation count

   Ex3: Operation count is proportional to $1+2+\ldots+n = O(n^2)$ (arithmetic sum)

   Ex4: $O(n)$ – version of Ex3, stores intermediate prefix sums

   Ex5: another n-element loop around the code from Ex3: $O(n^3)$ in total.

   Note: this is very inefficient. The two inner loops can be replaced by the code from Ex 4 at $O(n)$ cost and moved out of the outer loop, so total cost is $O(n)$!

2. Step 1: Sort the lists A, B, and C at $O(n \log n)$ total cost.

   Step 2: (in pseudo-code)

```
i, j, k := 0
while i ≤ len(A) - 1   AND   j ≤ len(B) - 1   AND   k ≤ len(C) - 1 :
    if A[i] == B[j] == C[k] :
        return False
    else increment the index where the corresponding list element is smallest
return True
```

   Note that step 2 sees at most $3n$ index increments, so it is $O(n)$.

3. Step 1: Take first 10 elements into a separate list M and sort them.
   This is $O(1)$ as the length of this list is fixed.

   Step 2: Go through rest of list, if element is encountered that is larger than smallest element of M, remove the latter and sort new element into M at $O(1)$ cost.

   ⇒ Total cost is $O(n)$

4. $f(n) = \begin{cases} n^2 & \text{if } n \text{ even} \\ 0 & \text{if } n \text{ odd} \end{cases}$

$n^2 > cn$

$n > c$

(i) Proof that $f(n)$ is not $O(n)$:

Let c be arbitrary. Then $\forall\, n > c$, n even, $f(n) > cn$.

$\Rightarrow \nexists\, n_0$ s.t. $\forall\, n \geq n_0 \quad f(n) \leq cn$.

(ii) Proof that $f(n)$ is not $\Omega(n)$:

Let c be arbitrary. Then $\forall\, n$, n odd, $f(n) < cn$

$\Rightarrow \nexists\, n_0$ s.t. $\forall\, n \geq n_0 \quad f(n) \geq cn$.

5. In Python:

```
def find_max (L):
    if len(L) == 1:
        return L[0]
    else return max (L[0], find_max L[1:])
```

If slices in Python are views, timing and memory are $O(n)$,
if they are copies, it's $O(n^2)$.

Note: $O(n)$ recursive implementation is easily possible!

6. See Python code in separate file.