# Exercises 2.2-1 Express the function n^3/1000 – 100n^2 + 3 in terms of Ѳ-notation.

As it’s mentioned in the chapter we ignore the lower-order terms and the leading term’s constant coefficient, so Ѳ(n^3).

## 2.2-2 Consider sorting n numbers stored in array A by first finding the smallest element of A and exchanging it with the element in A[1] . The find the second smallest element of A, and exchange it with A[2]. Continue in this manner for the first n – 1 elements of A. Write pseudocode for this algorithm, which is known as *selection sort*. What loop invariant does this algorithm maintain? Why does it need to run for only the first n – 1 elements, rather than for all n elements? Give the best-case and worst-case running time of selection sort in **Ѳ-**notation**.**

We start from first element of the array and for each position we search for the smallest number. Each position left from A[i] (A[i-1], A[i-2] and so on) the array is already sorted. We are terminating unil index reaches A.Length – 2 because at this position, the array is already sorted – in the last position is the biggest element (if we are sorting increasingly). The best case for this algorithm is Ѳ(n^2) because even if we take already sorted array, which is the best case, we have to iterate n-times to check if they are really sorted. The worst case is again Ѳ(n^2) as we have to go through all elements.

## 2.2-3 Consider linear search again. How many elements of the input sequence need to be checked on the average, assuming that the element being searched for is equally likely to be any element in the array? How about in the worst case? What are the average-case and worst-case running times of linear search in Ѳ-notation?

The best-case for Linear-search is the case when the element we are looking for is on the first position. In other words - Ѳ(1) which is a constant complexity. In other hand, the worst-case in when the element we are looking for is on the last position. Then we have to make n steps to find him - Ѳ(n).