Chapter 2 Part 2 Exercises

2.2-1

The function is:

$$\frac{n^3}{1000} - 100n^2 - 100n + 3$$

This can be equivalently written as:

$$c_1 n^3 - c_2 n^2 - c_3 n + c_4$$

where $c_1 = \frac{1}{1000}, c_2 = c_3 = 100$ and $c_4 = 3$.

Its Θ notation is $\Theta(n^3)$, because the higest term is n^3 and we ignore the factor c_1.

2.2-2

The selection-sort will be the following:

```
SELECTION-SORT($A$)
for $j = 1$ to $(A.lenght - 1)$
    $index = j$
    for $i = j$ to $A.length$
        if $A[index] > A[i]$
            $index = i$
    $temp = A[j]$
    $A[j] = A[index]$
    $A[index] = temp$
```

The **loop invariant** of the algorithm is:

At the start of each iteration of the outer loop, the subarray $A[1\,..\,j-1]$ is fully sorted.

Let us see now how the loop invariant properties hold now.

Initialization: We start by showing that the loop invariant holds before the first loop operation, when $j = 1$. The subarray of A is empty, so by definition it is sorted.

Maintenance: Informally, the body of the second for loop compares $A[index]$ with $A[i]$ and stores in $index$ the index of the smallest value in the range $A[j \mathinner{.\,.} n]$. The subarray $A[1 \mathinner{.\,.} j]$ consists of elements that are smaller or equal to $A[index]$, as otherwise $A[index]$ would have been chosen instead of them in the previous iteration.

Termination: We examine what happens when the loop terminates. When the loop terminates the value of $j = A.length - 1 = n - 1$. Then, the left subaray $A[1 \mathinner{.\,.} j - 1]$ is fully sorted based on the maintenance property, so the only item unsorted is the $A[A.length]$, which has to be bigger or equal than the previous item, as otherwise it would have been chosen in the previous step to be put in the $A[A.length - 1]$ position.

This algorith needs to run only for $n - 1$ elements, because during the final iteration $(n - 1)$, either the whole array is sorted or the last two elements, the n and $n - 1$, need to exchange positions. So, as the algorith in each iteration looks ahead and exchanges the element that belongs to that position when sorted with the one that was there beoforehand, there is no further elements to look ahead after the nth element, so it needs n-1 iterations.

In this algorithm, the best-case and worst-case running times are the same, as the inner loop will iterate through the non-sorted part of the array to find the next smallest element. So both the best-case and worst-case running time will be $\Theta(n^2)$.

2.2-3

The linear search algorithm is the one below:

```
LINEAR-SEARCH($A, v$)
while $i \leq A.length$
    if $A[i] == v$
        return $i$
    $i = i + 1$
return NIL
```

Averagely, over a number of runs of the algorithm, since the element being searched for is equally likely to be any element of the array, the average item would converge in the middle. This happens, because in the case that the element is in the end of the array all the elements will be searched (worst case) and in the case the element we are searching for is in the start of the array, only one element of the array needs to be checked (best case). That means that on average, half the elements would have to be searched to find the element we are looking for.

As mentioned, the worst case is when the element we are searching for is in the end of the array. Then, we would have to search for all the elements in the array.

It is easy to see from the instructions of the algorithm, that it's time complexity is linear to the number of the elements, so in the average case, $\frac{n}{2}$ iterations of instructions on line 1 and 2 would have to be made, meaning $\Theta(n)$ complexity.

In the worst case, the conditions in line 1 and 2 would have to be checked n times eached, leading to a $\Theta(n)$ complexity.

2.2-4

In order to have a good best-case time for every algorith, we should modify them to check if the condition of the best case is reached. That way we improve the speed of the algorith in best-case by a lot. For example, if in sorting algorithms we check wether the input is already sorted, their best-case running time would be improved.