

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:

$$c_1n^3 - c_2n^2 - c_3n + c_4$$

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

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

2.2-2

The selection-sort will be the following:

SELECTION-SORT(A)

```
1  for  $j = 1$  to ( $A.length - 1$ )
2       $index = j$ 
3      for  $i = j$  to  $A.length$ 
4          if  $A[index] > A[i]$ 
5               $index = i$ 
6   $temp = A[j]$ 
7   $A[j] = A[index]$ 
8   $A[index] = temp$ 
```

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

At the end of each iteration of the outer for loop, the subarray $A[1..j]$ is fully sorted.

This algorithm 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 algorithm in each iteration looks ahead and exchanges the element that belongs to that position when sorted with the one that was there beforehand, there is no further elements to look ahead after the n th 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( $A, v$ )
1  while  $i \leq A.length$ 
2      if  $A[i] == v$ 
3          return  $i$ 
4       $i = i + 1$ 
5  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 algorithm, we should modify them to check if the condition of the best case is reached. That way we improve the speed of the algorithm in best-case by a lot. For example, if in sorting algorithms

we check whether the input is already sorted, their best-case running time would be improved.