1. Rewrite the INSERTION-SORT procedure to sort sequences into nonincreasing instead of nondecreasing order.

2. Consider the **searching problem**:
Input: A sequence of n numbers $A =< a_1, a_2, \dots, a_n >$ and a value v.
Output: An index i such that $v = A[i]$ or the special value NIL if v does not appear in A.

   (a) Write pseudocode for **linear search**, which scans through the sequence, looking for the first occurrence of v.

   (b) Using a loop invariant, prove that your algorithm is correct. Make sure that your loop invariant fulfils the three necessary properties.

   (c) How many elements need to be checked on average, assuming that v is exactly one of the elements of A, and is equally likely to be in any position? Write running time in Θ-notation.

   (d) What is the worst-case running time in Θ-notation?

3. Consider sorting n numbers stored in array A by first finding the smallest element of A and exchanging it with the element in $A[1]$. Then 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.

   (a) Write pseudocode for this algorithm, which is known as **selection sort**.

   (b) What loop invariant does this algorithm maintain? Justify your answer.

   (c) Why does it need to run for only the first $n-1$ elements, rather than for all n elements?

   (d) Give the best-case and worst-case running times of this algorithm in Θ-notation.

4. We can express insertion sort as a recursive procedure as follows. In order to sort $A[1..n]$, we recursively sort $A[1..n-1]$ and then insert $A[n]$ into the sorted array. Write a recurrence for the worst-case running time of the recursive version of insertion sort.

5. **Correctness of Horner's rule**
The following code fragment implements Horner's rule for evaluating a polynomial

$$\begin{aligned} P(x) &= \Sigma_{k=0}^{n} a_k x^k \\ &= a_0 + x(a_1 + x(a_2 + \cdots + x(a_{n-1} + x a_n) \ldots)), \end{aligned}$$

given the coefficients $a_0, a_1, \ldots, a_n$ and a value for x:

```
1  y = 0
2  for i = n downto 0
3      y = a_i + x · y
```

(a) In terms of Θ-notation, what is the running time of this code fragment for Horner's rule?

(b) Write pseudoecode to implement the naive polynomial-evaluation algorithm that computes each term of the polynomial from scratch. What is the running time of this algorithm? How does it compare to Horner's rule?

(c) Consider the following loop invariant:

> At the start of each iteration of the **for** loop of lines 2-3,
> $y = \Sigma_{k=0}^{n-(i+1)} a_{k+i+1} x^k$.

Interpret a summation with no terms as equalling 0. Following the structure of a loop invariant proof, use this loop invariant to show that, at termination, $y = \Sigma_{k=0}^{n} a_k x^k$.

(d) Conclude by arguing that the given code fragment correctly evaluates a polynomial characterized by the coefficients $a_0, a_1, \ldots, a_n$.