ADw2

Name : **Peter Garamvoelgyi** Student ID: **2018280070**

1. **Write pseudocode for linear search, which scans through the sequence, looking for v. Using a loop invariant, prove that your algorithm is correct. Make sure that your loop invariant fulfills the three necessary properties. (CLRS 2.1-3)**

Input: A sequence of n numbers $A = \langle a_1, a_2, \ldots, a_n \rangle$ and a value v
Output: An index i s.t. $v = A[i]$ or the special value NIL if v does not appear in A.

```
LINEAR-SEARCH(A, v):
1  i := NIL
2  for j := 1 to A.length
3      if A[j] = v then
4          i := j
5  return i
```

The loop invariant I for this algorithm:

At the start of every iteration of the **for** loop, exactly one of the following conditions must be true:

a) $i = \text{NIL}$ and $A[1..j-1]$ does not contain v,
b) $i \neq \text{NIL}$ and $A[i] = v$.

Initialization:

At first, $j = 1$ and $i = \text{NIL}$. $A[1..0]$ is the empty array.
The empty array does not contain v.
a) is fulfilled, thus **I** is fulfilled.

Maintenance:

We know that at the beginning of every iteration **I** is fulfilled for $A[1..j-1]$. Let us consider four cases:

1. a) is fulfilled, i.e. $i = \text{NIL}$ and $A[1..j-1]$ does not contain v.
   1.1. If $A[j] = v$ then $i := j$. b) is fulfilled at the end of this iteration.
   1.2. If $A[j] \neq v$ then $A[1..j]$ does not contain v and $i = \text{NIL}$. a) is fulfilled at the end of this iteration.
2. b) is fulfilled, i.e. $i \neq \text{NIL}$ and $A[i] = v$.
   2.1. If $A[j] = v$ then $i := j$ and b) is fulfilled at the end of this iteration.
   2.2. If $A[j] \neq v$ then b) still remains fulfilled at the end of the iteration.

In all these possibilities, **I** remains fulfilled.

Termination:

At the end of the **for** loop we must have $j = n + 1$.
The invariant holds, i.e. either $A[1..n]$ does not contain v and $i = \text{NIL}$, or $i \neq \text{NIL}$ and $A[i] = v$. This proves that the algorithm is correct.

2. Consider linear search again (see Exercise 2.1-3). 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 Θ-notation? Justify your answers. (CLRS 2.2-3)

Let us have the following assumptions:

1. Each element in the array is unique.
2. The array contains v.
3. Our algorithm terminates right after finding v.

Let X be a random variable representing the number of elements we have to check until we find the first matching one. The expected value (corresponding to the average-case running time) is

$$f_{avg}(n) = E[X] = \sum_{k=1}^{n} k \cdot p_k = \sum_{k=1}^{n} k \cdot \frac{1}{n} = \frac{n+1}{2} = \Theta(n) \qquad n = A.length$$

To prove that this is indeed $\Theta(n)$, let us show the corresponding constants:

$$\forall\, n \geq 1: \quad 0 \leq \frac{1}{4} \cdot n \leq f_{avg}(n) \leq 1 \cdot n$$

The worst-case is of course the case where the array does not contain v, in which case the number of steps is n.

$$f_{worst}(n) = n = \Theta(n)$$

Again, we show the validity of the $\Theta(n)$-constraint by showing the constants:

$$\forall\, n \geq 1: \quad 0 \leq \frac{1}{2} \cdot n \leq f_{worst}(n) \leq 2 \cdot n$$

(Of course, these constants are just examples, there are infinitely many valid solutions.)

We can make our analysis more general by dropping assumption 2., i.e. we also regard the case when the array does not contain v. Let q denote the probability that the array contains v. This way, the expected value becomes:

$$E[X] = n \cdot (1-q) + \sum_{k=1}^{n} k \cdot p_k = n \cdot (1-q) + \sum_{k=1}^{n} k \cdot \frac{q}{n} =$$
$$= n \cdot (1-q) + \frac{q \cdot (n+1)}{2} = \Theta(n)$$

… which corresponds to our previous solution when $q = 1$.

Let us regard the case when we drop assumption 1. as well, i.e. the array elements do not have to be unique. Let Y be a random variable representing the position in the array that contains the value v. According to the problem statement

$$\forall\, k \in \{1,2,\dots,n\}: \quad P(Y = k) = p$$

… i.e. each position in the array contains the value v with equal probability. As the array can contain v multiple times, these events are independent. E.g.

$$\begin{array}{ll} P(Y = 1) & \sim A[1] = v \\ P(Y = 3) & \sim A[3] = v \\ P(Y = 1) \cap P(Y = 3) & \sim A[1] = v \ \wedge \ A[3] = v \end{array}$$

It is easy to see that

$$\forall\, i, j \in \{1,2,\dots,n\}, i \neq j: \quad P(Y = i, Y = j) = P(Y = i) \cdot P(Y = j)$$

Using this, the distribution of X corresponds to the geometric distribution

$$P(X = k) = (1-p)^{k-1} \cdot p; \qquad P(X \mid Y = \text{NIL}) = (1-p)^n$$

The expected value is given by

$$E[X] = n \cdot (1-p)^n + \sum_{k=1}^{n} k \cdot P(X = k) = n \cdot (1-p)^n + \sum_{k=1}^{n} k \cdot (1-p)^{k-1} \cdot p$$

3. Explain why the statement, "The running time of algorithm A is at least $O(n^2)$" is meaningless. (CLRS 3.1-3)

Def. $f(n) = \mathrm{O}(n^2)$**:** $\exists\, c, n_0 \in \mathbb{Z}^+\colon \quad \forall\, n \geq n_0\colon \quad 0 \leq f(n) \leq c \cdot n^2$

Intuitively, $\mathrm{O}(n^2)$ gives an (asymptotic) upper bound. A function cannot be larger than its upper bound.

Let $f(n)$ denote the running time of algorithm A for input size n.

$f(n) = O(n^2)$ means

$$\exists\, c, n_0 \in \mathbb{Z}^+\colon \quad \forall\, n \geq n_0\colon \quad 0 \leq f(n) \leq c \cdot n^2$$

At the same time, $f(n)$ is *at least* $O(n^2)$ suggests:

$$\exists\, c, n_1 \in \mathbb{Z}^+\colon \quad \forall\, n \geq n_1\colon \quad f(n) \geq c \cdot n^2$$

These two constraints can only be fulfilled simultaneously if

$$\exists\, c, n_2 \in \mathbb{Z}^+\colon \quad \forall\, n \geq n_2\colon \quad f(n) = c \cdot n^2$$

This is, however, a very special case (and this is just one interpretation of an otherwise rather ambiguous sentence). We can conclude that saying that an algorithm's running time is at least $\mathrm{O}(n^2)$ is meaningless.

4. Show that $2n^2 + 3n + 1 = \Theta(n^2)$.

Def. $f(n) = \Theta(n^2)$: $\exists\, c_1, c_2, n_0 \in \mathbb{Z}^+$: $\forall\, n \geq n_0$: $0 \leq c_1 \cdot n^2 \leq f(n) \leq c_2 \cdot n^2$

In our case:

$$f(n) = 2n^2 + 3n + 1$$

Our goal is to show three constants c_1, c_2, n_0 for which the definition above holds. Let us consider the inequalities

$$2n^2 + 3n + 1 \geq c_1 \cdot n^2 \quad (1)$$
$$2n^2 + 3n + 1 \leq c_2 \cdot n^2 \quad (2)$$

Rearranging, we get

$$(2 - c_1) \cdot n^2 + 3n + 1 \geq 0 \quad (1)$$
$$(2 - c_2) \cdot n^2 + 3n + 1 \leq 0 \quad (2)$$

Let $c_1 = 2$ and $c_2 = 6$[1]. The first inequality then becomes

$$3n + 1 \geq 0 \quad \rightarrow \quad n \geq -\frac{1}{3} \quad (1)$$

… i.e. (1) is fulfilled for any positive value n. The second inequality becomes

$$4n^2 - 3n - 1 = 4\left(n + \frac{1}{4}\right)(n - 1) \geq 0 \quad (2)$$

… i.e. we get a quadratic function with roots $n_1 = -1/4$ and $n_2 = 1$. The inequality is fulfilled when $n \leq n_1$ or $n \geq n_2$. For us, this means that our constraint is fulfilled for all $n \geq n_2$. From the above derivation, we get the constants

$$c_1 = 2 \quad c_2 = 6 \quad n_0 = 1$$

… for which

$$\forall\, n \geq 1: \quad 0 \leq 2n^2 \leq 2n^2 + 3n + 1 \leq 6n^2$$

Thus, $2n^2 + 3n + 1 = \Theta(n^2)$ holds.

[1] We choose these somewhat arbitrarily because we want to keep (1) linear and we want to make sure that the discriminant of (2) is an integer.