Introduction to Algorithms - Solutions

3rd Edition

Purbayan Chowdhury

October 2, 2020

Contents

Chapter 1

The Role of Algorithms in Computing

Chapter 2

Getting Started

2.1 Insertion Sort

Ex 2.1-1

Ex 2.1-2 Rewrite the INSERTION-SORT procedure to sort into non-increasing instead of non-decreasing order.

A.

Algorithm 1: Non-increasingInsertionSort

```
Input  : A ⟵ Unsorted Array
Output: A ⟵ Array Sorted in Non-increasing Order
for j ⟵ 1 to A.length − 1 do
    key ⟵ A[j]
    /* Insert A[j] into the sorted sequence A[1..j-1]      */
    i ⟵ j − 1
    while i ≥ 0 and A[i] > key do
        A[i + 1] ⟵ A[i]
        i ⟵ i − 1
    end
    A[i + 1] ⟵ key
end
```

Ex 2.2-3

Ex 2.2-4 Consider the searching problem: Input: A sequence of n numbers $A = (a_1, a_2, \ldots, a_n)$ Output: An index i such that $v = A[i]$ or the special value NIL if v does not appear in A. 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 fulfils the three necessary properties.

A.

Algorithm 2: Linear-Search

```
Input  : A ⟵ Array
         v ⟵ value to be searched
Output: i ⟵ index of the value if found, else NIL
i ⟵ NIL
for j ⟵ 0 to A.length − 1 do
    if A[j] = v then
        i ⟵ j
        break
    end
end
return i
```

Ex. 2.2-5 Consider the problem of adding two n-bit binary integers, stored in two n-element arrays A and B. The sum of the two integers should be stored in binary form in an $(n+1)$-element array C. State the problem formally and write pseudocode for adding the two integers.

A.

Algorithm 3: n-bitBinaryAddition

```
Input  : A ⟵ First Array
         B ⟵ Second Array
Output: C ⟵ Binary Addition Result
carry ⟵ 0
for i ⟵ n − 1 downto 0 do
    C[i + 1] ⟵ (A[i] + B[i] + carry)(mod2)
    if A[i] + B[i] + carry ≥ 2 then
        carry ⟵ 1
    end
    else
        carry ⟵ 0
    end
end
C[0] ⟵ carry
```

2.2 Analyzing algorithms

Ex 2.2-1

Ex 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 $A[1]$. 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. 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 times of selection sort in Θ-notation.

A.

Algorithm 4: SelectionSort

```
  Input  : A ⟵ Unsorted Array
  Output: A ⟵ Array Sorted in Increasing Order
1 for i ⟵ 0 to n − 1 do
2 |  min ⟵ i
3 |  for j ⟵ i + 1 to n do
  |  |  /* Find the index of the ith smallest element       */
4 |  |  if A[j] < A[min] then
5 |  |  |  min ⟵ j
6 |  |  end
7 |  end
8 |  Swap A[min] and A[i]
9 end
```

The loop invariant of selection sort is as follows:
At each iteration of the for loop of lines 1 through 9, the subarray $A[0 \ldots i-1]$ contains the $i-1$ smallest elements of A in increasing order. After $n-1$ iterations of the loop, the $n-1$ smallest elements of A are in the first $n-1$ positions of A in increasing order so the nth element is necessarily the largest amount.
The best-case and worst-case running times of selection sort are $\Theta(n^2)$, this is because regardless of how the elements are initially arranged, on the i-th iteration of the for loop in line 1, always inspects each of the remaining $n-i$ elements to find the smallest one remaining.
This yields a running
$\sum_{i=1}^{n-1} n - i = n(n-1) - \sum_{i=1}^{n-1} i = n^2 - n - \frac{n^2-n}{2} = \frac{n^2-n}{2} = \Theta(n^2)$

Chapter 3

Growth of Functions