Exercises

Giulia Bernardini
giulia.bernardini@units.it

Algorithmic Design / Algorithms for Scientific Computing
a.y. 2023/2024

Exercise 1

Cormen, problem 8.2: Suppose that we have an array A of n records to sort and that the key of each record has the value 0 or 1. An algorithm for sorting such a set of records might possess some subset of the following three desirable characteristics:

1. The algorithm runs in $O(n)$ time.
2. The algorithm is stable.
3. The algorithm sorts in place.

a. Give an algorithm that satisfies criteria 1 and 2 above.

b. Give an algorithm that satisfies criteria 1 and 3 above.

c. Give an algorithm that satisfies criteria 2 and 3 above.

Solution 1

a. Use an auxiliary array C of length n. Make two passes over A.

- In the first pass, copy all the records with key 0 in C, in the same order as they appear in A.
- In the second pass, do the same with the records with key 1, writing them in C right after the ones with key 0.
- Return C.

Homework: write pseudocode for this algorithm.

Solution 1

b. Scan A from left to right.

- Each time the algorithm finds a record with key 0, it swaps it with the record following the last 0 already seen.
- At the end of the scan, all the records with key 0 precede the records with key 1, thus A is sorted.
- The relative order of the records with the same key can change, thus it is not stable.

$index = 1$
for $i = 1$ to n **do**
    if $A[i] == 0$ **then**
        Swap $A[i]$ with $A[index]$
        $index = index + 1$
    end if
end for

Solution 1

c. Simply run Insertion Sort!

Exercise 2

1st problem of the written exam on 13/06/2022 (Prof. Casagrande's): Let A[1. . .n] be an array of integers containing at most k distinct values, where k is an unknown constant.

Propose an efficient in-place algorithm to sort A and establish its complexity.

Solution 2

Since k is constant, even an in-place algorithm can afford to use an auxiliary array D of size k to store the distinct items of A.

To compute D, we can use a list of size O(k). The list is initially empty. We scan A from left to right: whenever we read A[j], we check if it is already in the list; if not, we append it. This costs $\Theta(k^2)$ total time, which is $\Theta(1)$ because k is constant by hypothesis. We sort D using Insertion Sort in $\Theta(k^2) = \Theta(1)$ time.

We finally scan A k times. At the first scan, whenever we read A[j]=D[1], we move it at the beginning of A by swapping; at the d-th scan, we move items A[j]=D$[d]$ right after the elements moved in the $(d-1)$-th scan. Since D is sorted, at the end A will be sorted. This requires $\Theta(k^2 + kn) = \Theta(n)$ time.