CS21003 - Tutorial 3

August 21, 2018

1) 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.

2) Let $t \in N$ be a small constant and b a constant base (like 10 or 16). You are given an array of *n* integers, each in the range 0 *to* $b^t - 1$. Modify the radix sort algorithm to sort the input array in $O(n)$ time.

3) You are given an array of n dates in the $dd - mm - yyyy$ format. Propose a linear-time algorithm to sort the array in the usual increasing order (chronological order).

4) Suppose that n points are chosen uniformly inside a circle of radius r (that is, the probability of choosing a point in any region *R* of area a inside the circle is $a/(\pi r^2)$. Give an algorithm that sorts the *n* given points with respect to their distances from the center of the circle in expected linear time.

5) Suppose that you are given a sequence of *n* elements to sort. The input sequence consists of *n/k* subsequences, each containing *k* elements. The elements in a given subsequence are all smaller than the elements in the succeeding subsequence and larger than the elements in the preceding subsequence. Thus, all that is needed to sort the whole sequence of length *n* is to sort the *k* elements in each of the *n/k* subsequences. Show an $\Omega(n \lg k)$ lower bound on the number of comparisons needed to solve this variant of the sorting problem. (Hint: It is not rigorous to simply combine the lower bounds for the individual subsequences.)

6) Suppose that you are given n red and n blue water jugs, all of different shapes and sizes. All red jugs hold different amounts of water, as do the blue ones. Moreover, for every red jug, there is a blue jug that holds the same amount of water, and vice Versa. Your task is to find a grouping of the jugs into pairs of red and blue jugs that hold the same amount of water. To do so, you may perform the following operation: pick a pair of jugs in which one is red and one is blue, fill the red jug with water, and then pour the water into the blue jug. This operation will tell you whether the red or the blue jug can hold more water, or that they have the same volume. Assume that such a comparison takes one time unit. Your goal is to find an algorithm that makes a minimum number of comparisons to determine the grouping. Remember that you may not directly compare two red jugs or two blue jugs.

   a) Describe a deterministic algorithm that uses $\Theta(n^2)$ comparisons to group the jugs into pairs

   b) Give an algorithm whose expected number of comparisons is $O(n \lg n)$. What is the worst-case number of comparisons for your algorithm?

7) The QUICKSORT algorithm contains two recursive calls to itself. After QUICKSORT calls PARTITION , it recursively sorts the left subarray and then it recursively sorts the right subarray. The second recursive call in QUICKSORT is not really necessary; we can avoid it by using an iterative control structure. This technique, called **tail recursion**, is provided automatically by good compilers. Consider the following version of quicksort, which simulates tail recursion:

```
TAIL-RECURSIVE-QUICKSORT($A, p, r$)
while $p < r$
    // Partition and sort left subarray.
    $q$ = PARTITION($A, p, r$)
    TAIL-RECURSIVE-QUICKSORT($A, p, q - 1$)
    $p = q + 1$
```

a. Argue that TAIL-RECURSIVE-QUICKSORT(A, 1, A.length) correctly sorts the array A. Compilers usually execute recursive procedures by using a stack that contains pertinent information, including the parameter values, for each recursive call. The information for the most recent call is at the top of the stack, and the information for the initial call is at the bottom. Upon calling a procedure, its information is pushed onto the stack; when it terminates, its information is popped. Since we assume that array parameters are represented by pointers, the information for each procedure call on the stack requires O(1) stack space. The stack depth is the maximum amount of stack space used at any time during a computation.

b. Describe a scenario in which TAIL-RECURSIVE-QUICKSORT's stack depth is *Θ(n)* on an n-element input array

C. Modify the code for TAIL-RECURSIVE-QUICKSORT so that the worst-case stack depth is *Θ(lg n)*. Maintain the *O(n lg n)* expected running time of the algorithm.

```
QUICKSORT($A, p, r$)
if $p < r$
    $q$ = PARTITION($A, p, r$)
    QUICKSORT($A, p, q - 1$)
    QUICKSORT($A, q + 1, r$)
```

```
PARTITION($A, p, r$)
$x = A[r]$
$i = p - 1$
for $j = p$ to $r - 1$
    if $A[j] \le x$
        $i = i + 1$
        exchange $A[i]$ with $A[j]$
exchange $A[i + 1]$ with $A[r]$
return $i + 1$
```