Fundamental Algorithms

Homework 2

Due: June 14th, 5:30 PM EST

Instructions

Please answer each **Problem** on a separate page. Submissions must be uploaded to your account on Gradescope by the due date and time above.

Note

You have to wait until the lecture on June 9th to solve some of the problems.

Problems to submit

Problem 1 (10+15 points)

Recall that MERGE_SORT splits the input array into two halves of almost equal sizes, and after each half is sorted recursively, both of them are merged into a sorted array.
Let's now consider a variant of MERGE_SORT, where instead of splitting the array into two equal size parts, we split it into four parts of almost equal sizes, recursively sort out each part, and at the end, merge all four parts together into a sorted final array.
In what follows, we want to find the time complexity of this variant of MERGE_SORT.

(a) Define $T(n)$ as the running time of this variant on any input of size n. Find a recursive formula for $T(n)$, i.e., write $T(n)$ in terms of $T(k)$ for some $1 \leq k < n$.

(b) Draw the corresponding recursion tree and find the height of the tree, the running time of each layer, and the sum of running times for all layers. Then use this information to find the explicit answer for $T(n)$.
You may assume that n is a power of 4, i.e., $n = 4^k$ for some positive integer k.
For full credit, your answer must use $\Theta(.)$ notation.

Problem 2 (10+15 points)

Let A be a sorted array with n distinct elements. Consider the following algorithm that finds an integer $i \in \{j \dots k\}$ such that $A[i] = x$, or returns **FALSE** if no such integer i exists.

```
1 FIND($A, j, k, x$)
2     If $j > k$ Return FALSE
3     Define $i := \dots$
4     If $A[i] = \dots$ Return $\dots$
5     If $A[i] < \dots$ Return FIND($\dots$)
6     Return FIND($\dots$)
```

(a) Fill in the blanks to complete the algorithm.

(b) Write the recurrence for the running time of the algorithm. Draw the corresponding recursion tree and find the explicit answer to the recursion.
You may assume that n is a power of 2, i.e., $n = 2^m$ for some positive integer m.
Your answer must use $\Theta(.)$ notation.

Problem 3 (5+20+5 points)

Given an array $A[1, 2, \ldots, n]$, recall that the PARTITION function in QUICKSORT takes the last element $A[n]$ as the pivot and partitions A into two parts: the left part consisting of the elements which are less than or equal to the pivot and the right part consisting of the rest.

(a) Assume that the elements in A are all the same. What is the running time of QUICKSORT on A? Explain your answer.

You should be convinced by part (a) that the running time is not good! To overcome this issue, we modify the PARTITION function so that after running it, any input array A is partitioned into three parts: the left part consisting of all elements strictly less than the pivot, the middle part consisting of all elements equal to the pivot, and the right part consisting of the rest (i.e., all elements strictly larger than the pivot).

(b) Write down the pseudo-code for this modified version of the PARTITION function. Note that your algorithm must be an in-place algorithm (no auxiliary array should be used) with $\Theta(n)$ time complexity.

Hint: Define three indices: i as the end of the left part, k as the end of the middle part, and j denoting the current index which is used in the for-loop. Modify the PARTITION function discussed in the lecture for building both the left part and the middle part iteratively.

(c) Use the modified version of the PARTITION function obtained in part (b) in the QUICKSORT algorithm and derive a modified QUICKSORT algorithm. Let A be an array consisting of n identical elements. What is the running time of this version of QUICKSORT on A? Explain your answer.

Problem 4 (20 points)

Let A be an array of n integers, where n is divisible by 4. Describe a randomized algorithm with $O(n)$ time complexity to decide whether there exists an element that occurs more than $n/4$ times in A.

(a) Show that if such an element exists in A, it must be either the $(n/4)$-th, or the $(2n/4)$-th, or the $(3n/4)$-th smallest elements.

(b) Use part (a) to derive your algorithm.

Bonus problem

The following problems are optional. They will NOT be graded and they will NOT appear on any of the exams. Work on them only if you are interested. They will help you to develop problem solving skills for your future endeavors.

Bonus Problem 1

Given an array $A[1 \ldots n]$ consisting of n distinct elements, we want to find the maximum and the second maximum elements among the elements of A.

(a) Design an algorithm which uses at most $2n - 3$ comparisons.

(b) What is the minimum number of comparisons you can use?

(c) Can you use at most $n + \log_2 n - 2$ comparisons?