[redacted]

Problem Set 5

[redacted] *Due: 8 am Thursday, October 14*

Problem 5-1 (Merging Lists) 17+6 points

In the previous problem set, you presented an algorithm that ran in time $\Theta(n \log k)$, merging k sorted lists, each containing at least one item. In this problem, we will begin by trying to merge k such sorted lists, without the use of a heap. We will then revisit the simpler case of $k = 2$ and prove that the algorithm MERGE is indeed asymptotically optimal, i.e., meets the lower-bound for the running time of the algorithm.

(a) (5 points) You are given $k \geq 1$ sorted arrays $A_0, \ldots, A_{k-1}$. Array A_i has length $n_i \geq 1$ for $0 \leq i \leq k-1$ with $\sum_{i=1}^{k-1} n_i = n$. You may assume that all the elements are distinct. Present a recursive algorithm MULTIWAY-MERGE$(A_i, \ldots, A_j, n_i, \ldots, n_j)$ that returns the final sorted array A and its size n. Your algorithm will not use a heap. Instead, it can only invoke the procedure MERGE(X, m_1, Y, m_2) and returns Z, m. Here, $X[1, \ldots, m_1]$ and $Y[1, \ldots, m_2]$ are two sorted arrays and $Z[1, \ldots, m]$ is the merged version of these two sorted lists with $m = m_1 + m_2$. This algorithm runs in time $\Theta(m_1 + m_2)$. You may skip the proof of correctness in your solution but try to prove, to yourself, that your algorithm is correct. (**Hint**: Do not forget to state the invocation call for your algorithm, i.e., that first call which is included in the main/driver function which triggers the algorithm.)

(b) (3 points) For analyzing the running time, it is useful to simplify the notation. We will assume that each array A_i is of length ℓ. In other words, $n = k\ell$. Now, let $T(k)$ denote the running time of MULTIWAY-MERGE when run on k sorted arrays, each with ℓ elements. Now, derive a recurrence relation of $T(k)$. Then, solve it (by any method of your choosing), to get that $T(k) = \Theta(k\ell \cdot \log(k))$ for $k \geq 2$ (We usually write it as $\Theta(k\ell \cdot \log(1 + k))$ to also handle the case when $k = 1$ but that is just an aside). (**Hint**: Do not forget to include base cases for your recurrence relation. Also, observe the occurrence of ℓ in the final answer. Therefore, your recurrence relation will also contain ℓ but is a constant.)

Now, let us take a closer look at the algorithm MERGE. Note that MERGE, when merging two arrays with a total of n elements, runs in time $\Theta(n)$. This algorithm makes $n - 1$ comparisons in total. We will prove that it is indeed optimal, i.e., we have two individually sorted arrays A_0, A_1 each with $n/2$ elements for a total of n elements.

(c) (6 points) Use the decision tree method to prove that any comparison-based 2-way merging algorithm makes at least $n - o(n)$ comparisons. (**Hint**: Start with proving that that the number of possible leaves of the tree is equal to the number of ways to partition an n element array into 2 sorted lists of size $n/2$, and then compute the latter number. You may use Stirling's Approximation, i.e., $a! \approx \sqrt{2\pi a} \cdot (a/e)^a$.)

(d) (**Extra Credit**)(6 points) In the previous part, you proved that any algorithm that relies on comparisons to merge two sorted lists, each containing $n/2$ elements, makes at least $n - o(n)$ comparisons. However, we know that MERGE actually makes $n - 1$ comparisons in total. In this part, you will improve your lower bound from part (c).

Let $A = [a_1, \ldots, a_{n/2}]$ and $B = [b_1, \ldots, b_{n/2}]$ are the input sorted arrays of distinct integers and $C[1, \ldots, n]$ be the final sorted output. Let $a_i < b_j$ be consecutive elements in C. Then, show that any path of execution from the root to the leaf will compare (a_i, b_j). Use this to improve your lower bound.

(**Hint**: Use a similar approach to what was seen in the element uniqueness decision tree proof. Define a new array B', prove that it is sorted, and then conclude the proof.)

(e) (3 points) Let us go back to our original problem of merging k sorted lists. We have looked at two solutions, both of which run in time $O(n \log k)$. Someone claims to have an algorithm that can merge k sorted lists, through only comparisons, in time $O(n \log \log k)$. Disprove this claim by relying on the sorting lower-bound.

Problem 5-2 (Running Median) 14+6 points

In this task you will design a data structure supporting the following operations:

- BUILD($A[1 \ldots n]$): Initializes the data structure with the elements of the (possibly unsorted) array A with duplicates.
- INSERT(x): Inserts the element x into the data structure.
- MEDIAN: Returns the median[1] of the currently stored elements.

In the following, you are allowed to use the standard heap operations for both Min and Max Heap as defined in the textbook/lectures and incur the corresponding cost. The goal of this problem is to design a data structure that achieves the following running-time for the above functions:

- BUILD runs in time $O(n)$
- INSERT runs in time $O(\log n)$
- MEDIAN runs in time $O(1)$

(a) (2 points) Present a data structure that you will use to achieve the required operations in the desired running time. You will be asked to provide implementations of these algorithms in later parts. For this part, simply describe your data structure, i.e., provide an invariant (not a loop invariant) that you will maintain in this data structure.
(**Hint**: Use more than one heap.)

(b) (**Extra Credit**)(6 points) Consider an array that contains many duplicates and observe that for such an array, quicksort recurses on all duplicates of the pivot element. In this task you are to develop a new partitioning procedure that works well on arrays with many duplicates.

[1] The median of n elements is the $\lceil \frac{n}{2} \rceil$-smallest element.

The idea is to partition the array into elements less than the pivot, equal to the pivot and greater than the pivot. Present the algorithm that does this three-way partitioning called 3-WAY-PARTITION(A, p, r). The algorithm will modify A in-place and return two indices i, j such that $A[p, \ldots, i-1]$ contains element strictly less than the pivot, $A[i, \ldots, j-1]$ contains element equal to the pivot, and $A[j, \ldots, r]$ contains elements strictly greater than the pivot. This algorithm should run in linear-time. Ensure that your algorithm is in-place, i.e., does not require more than a constant amount of extra space.

Do not forget to briefly argue the correctness of your algorithm and justify the running time of your algorithm.

Now, we look at the pseudocode implementations of the three algorithms that we desire. For these parts, you may assume the existence of *a correct* in-place, linear-time, 3-WAY-PARTITION(A, p, r) algorithm (even if you did not complete part (b)).

(c) (5 points) Present the pseudocode for BUILD$(A[1, \ldots, n])$. Justify the correctness and running time of your algorithm. You will use $O(n)$ additional space for this problem. (**Hint**: Remember duplicates may be present and need to be handled carefully.)

(d) (5 points) Present the pseudocode for INSERT(x). Justify the correctness and running time of your algorithm.

(e) (2 points) Present the pseudocode for MEDIAN(). Justify the correctness and running time of your algorithm.

Problem 5-3 (Running Median 2) 10 points

In the previous problem, we designed a data structure supporting the following operations:

- BUILD$(A[1 \ldots n])$: Initializes the data structure with the elements of the array A in time $O(n)$.
- INSERT(x): Inserts the element x into the data structure in time $O(\log n)$, where n is the number of elements stored in the data structure.
- MEDIAN: Returns the median[2] in time $O(1)$ of the currently stored elements.

In this problem we assume that all elements are integer numbers from 1 to 10^6, and we will design a data structure that supports the same set of operations but improves on the insertion process - it should insert an element in constant time.

(a) (3 points) Present a data structure that you will use to achieve the required operations in the desired running time. You will be asked to provide implementations of these algorithms in later parts. For this part, simply describe your data structure, i.e., provide an invariant (not a loop invariant) that you will maintain in this data structure.

In parts (b)-(c), describe in pseudo-code your implementations from part (a) and analyze the running time.

[2]The median of n elements is the $\lceil \frac{n}{2} \rceil$-largest element.

(b) (4 points) Present the pseudocodes for BUILD($A[1, \ldots, n]$) and INSERT(x). Justify the correctness and running time of your algorithms.

(c) (3 points) Present the pseudocode for MEDIAN() that runs in constant time. Justify the correctness and running time of your algorithm