Problem 4-1 (Fast $k$-way Merging) 10 points

Using a min-heap in a clever way, give $O(n \log k)$-time algorithm to merge $k$ sorted arrays $A_1 \ldots A_k$ of size $n/k$ each into one sorted array $B$. Write the pseudocode of your algorithm using procedures \textsc{Build-Heap}, \textsc{Extract-Min} and \textsc{Insert}.

Problem 4-2 (Finding a Repetitor) 26 (+14) Points

Your are given an array $A[1] \ldots A[n]$ of $n$ “objects”. You have a magic unit-time procedure \textit{Equal}(A[i], A[j]), which will tell if objects $A[i]$ and $A[j]$ are the same. Unfortunately, there is no other way to get any meaningful information about the objects: e.g., cannot ask if $A[i]$ is “greater” than $A[j]$ of if it is more “sexy”, etc., just the equality test. We say that $A$ is a repetitor if it contains strictly more than $n/2$ elements which are all pairwise the same. In this case any of $A$’s (at least $n/2$) repetitive elements is called dull. For example, if the “object” is a string, the array (boring, funny, cute, boring, boring) is a repetitor where boring is dull while funny is not. On the other hand, the array (hello, hi, bonjorno, hola, whasup) is not a repetitor. You goal is to determine if $A$ is a repetitor, and, if so, output its dull “object” (which is clearly unique).

(a) (8 points) Design a simple divide-and-conquer algorithm for this problem running in time $O(n \log n)$. Make sure you argue the correctness and the running time.
(Hint: Prove that if $A$ is a repetitor, at least one of its “halves” is as well.)

(b) (4 Points) Remember, if $A$ was an integer array, the procedure \textsc{Partition}(A, p, r) (see Section 7.1) makes $x = A[r]$ the pivot element and returns the index $q$, where the new value of $A[q]$ contains the pivot $x$, the new values $A[p \ldots q - 1]$ contain elements less or equal to $x$, and the new values $A[q + 1 \ldots r]$ contain values greater than $x$. Write the pseudocode of the modified procedure \textsc{New-Partition}(A, p, r), which only uses the \textit{Equal} operator and returns $q$ such that $A[q + 1 \ldots r]$ contain all the elements equal to $x$ (while $A[p \ldots q]$ contain all other elements).

(c) (2 points) Consider the following, more general, algorithm \textsc{Repeat}(A, n, t), which tells if some element of $A[1] \ldots A[n]$ is repeated at least $t$ times. (Clearly, \textsc{Repitor} can just call \textsc{Repeat} with $t = n/2 + 1$.)

\begin{verbatim}
Repeat(A, n, t)
  If n < t Return no
  Pick i ∈ {1 .. n} at random.
  Swap(A[i], A[n])
\end{verbatim}
\[ q \leftarrow \text{New-Partition}(A, 1, n) \]
\[ \text{If } n - q \geq t \text{ Then Return } (\text{yes}, A[n]) \]
\[ \text{Return } \text{Repeat}(A, q, t) \]

Argue that the algorithm above is correct.

(d) (3 points) Argue that the algorithm above always terminates in time \(O(n^2)\) (irrespective of the random choices of \(i\)).

(e) (3 points) Give an example of an (integer) array \(A\) and a value \(t \geq 2\) where the algorithm indeed takes time \(\Omega(n^2)\).

(f) (4 Points) Let \(T(n)\) be the worst case (over all arrays \(A[1\ldots n]\) and \(t > n/2\) such that \(A\) contains \(t\) identical elements) of the expected running time of \(\text{Repeat}(A, n, t)\) (over the random choice of \(i\)). For concreteness, assume \(\text{New-Partition}\) takes time exactly \(n\). Prove that
\[ T(n) \leq \frac{1}{2} \cdot T(n-1) + n \]

(Hint: Prove that in this case no recursive sub-call will be made with probability \(t/n > 1/2\).)

(g) (2 points) Show by induction that \(T(n) \leq 2n\).

(h\*) (Extra Credit; 6 points) Consider the following test for repetitor. For 100 times, run \(\text{Repeat}(A, n, n/2 + 1)\) for at most \(4n\) steps. If one of these 100 runs ever finishes within \(4n\) steps, use that answer. If none of the 100 runs terminates within \(4n\) steps, return \text{no}. Argue that the running time of this procedure is \(O(n)\). Then argue that the probability it returns the incorrect \text{no} answer (when it should have returned \text{yes}) is at most \(2^{-100}\).

(Hint: Show that when the answer is \text{yes}, the probability of not finding this answer in \(4n\) steps is at most \(1/2\). Google for “Markov’s inequality” if you want to be formal.)

(i\**) (Extra Credit; 8 points) Try to design \(O(n)\) deterministic test for a repetitor.

**Problem 4-3 (QuickSort)** 12 points

Assume we are given an array \(A[1\ldots n]\) of \(n\) distinct integers and that \(n = 2k\) is even.

(a) (4 points) Let \(\text{pivot}(A)\) denote the rank of the pivot element at the end of the partition procedure, and assume that we choose a random element \(A[i]\) as a pivot, so that \(\text{pivot}(A) = i\) with probability \(1/n\), for all \(i\). Let \(\text{smallest}(A)\) be the length of the smaller sub-array in the two recursive subcalls of the \text{QuickSort}. Notice, \(\text{smallest}(A) = \min(\text{pivot}(A) - 1, n - \text{pivot}(A))\) and belongs to \(\{0\ldots k-1\}\), since \(n = 2k\) is even. Given \(0 \leq j \leq k-1\), what is the probability that \(\text{smallest}(A) = j\)?

(b) (3 points) Compute the expected value of \(\text{smallest}(A)\); i.e., \(\sum_{j=0}^{k-1} \Pr(\text{smallest}(A) = j) \cdot j\).

(Hint: If you solve part (a) correctly, no big computation is needed here.)

(c) (5 points) Write a recurrence equation for the running time \(T(n)\) of \text{QuickSort}, assuming that at every level of the recursion the corresponding sub-arrays of \(A\) are partitioned exactly in the ratio you computed in part (b). Solve the resulting recurrence equation. Is it still as good as the average case of randomized \text{QuickSort}?
Problem 4-4 (The $k$ Smallest Elements)  12 points

We wish to implement a data structure $D$ that maintains the $k$ smallest elements of an array $A$. The data structure should allow the following procedures:

- $D \leftarrow \text{initialize}(A, n, k)$ that initializes $D$ for a given array $A$ of $n$ elements.
- $\text{traverse}(D)$, that returns the $k$ smallest elements of $A$ in sorted order.
- $\text{insert}(D, x)$, that updates $D$ when an element $x$ is inserted in the array $A$.

We can implement $D$ using one of the following data structures: (i) an unsorted array of size $k$; (ii) a sorted array of size $k$; (iii) a max-heap of size $k$.

(a) (4 points) For each of the choices (i)-(iii), show that the \text{initialize} procedure can be performed in time $O(n + k \log n)$.

(b) (3 points) For each of the choices (i)-(iii), compute the best running time for the \text{traverse} procedure you can think of. (In particular, tell your procedure.)

(c) (5 points) For each of the choices (i)-(iii), compute the best running time for the \text{insert} procedure you can think of. (In particular, tell your procedure.)