Problem 5-1 (Sorting in $n \log \log n$ Time)  

(a) (6 points) Assume you are given an array of $n$ integers with many duplications, so that you know that there are at most $\log n$ distinct elements in the array. Show how to sort this array in time $O(n \log \log n)$.

(b) (4 points) Assume you are given an array of $n$ integers in the range $\{1, \ldots, (\log n)^{\log n}\}$. Show how to sort this array in time $O(n \log \log n)$.

Problem 5-2 (Choosing the Right Tool)  

For each example choose one of the following sorting algorithms and carefully justify your choice: HEAPSORT, RADIXSORT, COUNTINGSORT. Give the expected runtime for your choice as precisely as possible. If you choose RADIX SORT then give a concrete choice for the basis (i.e. the value of “$r$” in the book) and justify it. (Hint: We assume that the array itself is stored in memory, so before choosing the fastest algorithm, make sure you have the space to run it!)

(a) Sort the length $2^{16}$ array $A$ of 128-bit integers on a device with 100MB of RAM.

(b) Sort the length $2^{24}$ array $A$ of 256-bit integers on a device with 600MB of RAM.

(c) Sort the length $2^{16}$ array $A$ of 16-bit integers on a device with 1GB of RAM.

Problem 5-3 (Lower Bound for Min-Element)  

Recall that there exists a trivial algorithm for searching for a minimal element of an array of size $n$ that needs exactly $(n - 1)$ comparisons. The purpose of this problem is to prove that this is optimal solution.

You can assume that elements of the input array are distinct. As usual, you can represent any comparison-based algorithm for Min-Element problem as a binary decision tree, whose internal nodes are labeled by $(i : j)$ (meaning “compare $A[i]$ and $A[j]$”), and a left child is chosen if and only if $A[i] < A[j]$ (recall, we assume distinct elements). And the leaves are labeled by the index $i$ meaning that the smallest element in the arrange is $A[i]$.

(a) (4 points warm-up) Show that the naive “counting leaves” application of decisional tree method (ala sorting lower bound) will only give a very suboptimal lower bound $\Omega(\log n)$ for the Min-Element problem.
(b) (4 points) Here we will try to enrich the decisional tree by adding some additional info $S(v)$ to every vertex $v$. Precisely, $S(v)$ is the set of all indices $i$ which are “consistent” with all the comparisons made from the root to $v$ (excluding $v$ itself), where “consistent” means that there exists an array $A$ whose minimum is $A[i]$ and the node $v$ is reached on input $i$. For example, if $v$ is any (reachable) leaf labeled by $i$, then $S(v) = \{i\}$ (as otherwise the algorithm would not be correct). Assuming the root node $root$ is labeled ($i : j$), describe $S(root)$, $S(root.left)$ and $S(root.right)$.

(Hint: What element is impossible to be minimal if you know that $A[i] < A[j]$?)

(c) (4 points) Generalize part (b): show that for every vertex $v$ we have: $S(v.left) = S(v) \setminus \{\ldots\}$ and $S(v.right) = S(v) \setminus \{\ldots\}$ (you need to fill dots on your own).

(d) (4 points) Use (c) to prove that in any valid decision tree for Min-Element, any leaf (which is stronger than some leaf) must have depth at least $(n − 1)$. (Hint: Think about $|S(v)|$ as $v$ goes from root to this leaf.)

Problem 5-4 (Roller coaster) 6 points


Give a linear algorithm that any given array $A$ with $n$ distinct elements transforms into a roller coaster array $B$. Namely, $B$ must contain exactly the same $n$ distinct elements as $A$, but must also be a roller coaster. (Hint: A median may be useful here.)