Problem 5-1 (Missing number) 8 pts

Given an unsorted array $A[1\ldots n]$ which contains all but one of the integers in the range $0\ldots n$, we want to determine the missing integer. Consider the following algorithm:

\[
\text{Missing}(A, n) \\
\text{allocate new arrays } B, C \text{ of size } n + 1 \\
pick \text{random } x \text{ from } 0, \ldots, n \quad // \text{assume this step takes time 1} \\
n_{\text{less}} = 0, n_{\text{gr}} = 0 \\
\text{For } i = 1 \text{ to } n \\
\quad \text{If } A[i] < x \\
\quad \quad n_{\text{less}} ++ \\
\quad \quad B[n_{\text{less}}] = A[i] \\
\quad \text{Else If } A[i] > x \\
\quad \quad n_{\text{gr}} ++ \\
\quad \quad C[n_{\text{gr}}] = A[i] - x - 1 \\
\text{If } n_{\text{less}} + n_{\text{gr}} = n \\
\quad \text{Return } x \\
\text{Else If } n_{\text{less}} < x \\
\quad \text{Return } \text{Missing}(B, n_{\text{less}}) \\
\text{Else Return } \text{Missing}(C, n_{\text{gr}}) + x + 1
\]

(a) (2 pts) Prove the correctness of the algorithm.

(b) (2 pts) Let $T(n)$ be the expected running time. Write the recurrence equation for $T(n)$. (Hint: equation is similar to the one for Randomized Select.)

(c) (2 pts) Solve for $T(n)$. (Hint: use similar techniques)

(d) (2 pts) Design your own best deterministic algorithm for the missing number problem, and asymptotically compare its running time with the answer in part (c).

Problem 5-2 (QuickSort vs Insertion Sort) 10 Points

We say that an array $A$ is $c$-nice, where $c$ is a constant (think 100), if for all $1 \leq i, j \leq n$, such that $j - i \geq c$, we have that $A[i] \leq A[j]$. For example, 1-nice array is already sorted. In this problem we will sort such $c$-nice $A$ using INSERTION SORT and QUICKSORT, and compare the results.
(a) (4 Points) In asymptotic notation (remember, $c$ is a constant) what is the worst-case running time of INSERTION SORT on a $c$-nice array? Be sure to justify your answer.

(b) (5 Points) In asymptotic notation (remember, $c$ is a constant) what is the worst-case running time of QUICKSORT on a $c$-nice array? Be sure to justify your answer.

(c) (1 Points) Which algorithm would you prefer?

**Problem 5-3 (Improving Deterministic Quicksort)** 16 points

We give the following procedure STRANGE SORT to sort an array $A$ of $n$ distinct elements.

- Divide $A$ into $n/3$ subsets of size 3 each
- Compute the median of each set to get $n/3$ elements
- Sort these $n/3$ elements recursively using STRANGE SORT and get $p$ as the median of these $n/3$ elements
- Use $p$ as a pivot in the PARTITION procedure of QUICKSORT, i.e., divide $A$ into sets with elements less than $p$ (call $A_{<p}$), those with elements equal to $p$, and those with elements greater than $p$ (call $A_{>p}$)
- Recursively call STRANGE SORT on $A_{<p}$ and $A_{>p}$.

(a) (4 points) Give a lower bound on the number of elements in $A_{<p}$ and $A_{>p}$.

(b) (2 points) What is the recurrence relation you obtain for the time complexity $T(n)$ of STRANGE SORT assuming that the position of $p$ corresponds to the lower bound found in part (a)?

(c) (4 points) Try to use induction to show that $T(n) \leq n^{1+c}$ for some (yet undetermined) value of $c > 0$. For simplicity, you may ignore the term corresponding to the (non-recursive) divide-and-conquer step in your recurrence relation, when you do your inductive step. (In other words, only keep all the “$T$-terms” and drop the “$f(n)$-terms when you prove your inductive step.) What is the inequality that $c$ should satisfy in order for the induction to work?

(d) (2 points) Use http://www.wolframalpha.com to find the smallest possible $c$ (upto two decimal places) for which the “nasty” inequality in part (c) is satisfied. How does the resulting running time of STRANGE SORT compare to the worst and average-case running times of QUICKSORT?

(e) (4 points) How does your answer from part (d) change if you replace sets of size 3 by sets of size 5 in the STRANGE SORT algorithm.

---

1 It is easy to see that this is the worst case, but you do not have to show this.

2 I.e., the $n/3$ median findings on 3 elements and the run of the PARTITION procedure around $p$. 
Problem 5-4 (Fast $k$-way Merging)  

Consider the problem of merging $k$ sorted arrays $A_1, \ldots, A_k$ of size $n/k$ each, where $k \geq 2$.  

(a) (8 points) Using a min-heap in a clever way, give an $O(n \log k)$-time algorithm to solve this problem. Write the pseudocode of your algorithm using procedures BUILD-HEAP, EXTRACT-MIN and INSERT.  

(b) (8 points) Let the number of arrays be $k = 2$. Assume all $n$ numbers are distinct. Using the decision tree method and the fact (which you can assume without proof) that $\binom{n}{n/2} \approx \frac{2^n}{\Theta(\sqrt{n})}$, show that the number of comparisons for any comparison-based 2-way merging is at least $n - O(\log n)$.  

(Hint: Start with proving 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.)  

(c) (2 pts) Show that any correct comparison-based 2-way merging algorithm must compare any two consecutive elements $a_1$ and $a_2$ in merged array $B$, where $a_1 \in A_1$ and $a_2 \in A_2$. Use this fact to construct an instance of 2-way merging which requires at least $n - 1$ comparisons, improving your bound of part (b).  

(d*) (6 pts) Extra Credit: Show that for general $k$, any comparison-based $k$-way merging must take $\Omega(n \log k)$ comparisons, showing that your solution to part (a) is asymptotically optimal.  

(Hint: You can either try to extend part (b) (easier) or part (c) from $k = 2$ to general $k$. Beware that calculations might get messy...)  

PS5, Page 3