Mathematics 5365 (Analysis of Algorithms)

Midterm 1

You may use the randomized binary search tree algorithms without reimplementing them. There is no guarantee that any problem can benefit from these, though.

1. Input data: $S\colon \mathsf{Order}$, $n\colon \mathbf{N}$, $x\colon S[n]_{\le}$, $s\colon S$, $s \ne x[i]$ for all $i \in [0, n)$. Output data: $a\colon \mathbf{N}$ such that $a \in [0, n]$, $x[i] < s$ for all $i \in [0, a)$, and $x[i] > s$ for all $i \in [a, n)$. Requirements: the average number of comparisons performed by the algorithm must be as small as possible. Each of the $n + 1$ values of a is equally likely to occur (with probability $1/(n + 1)$).

2. Input data: $S\colon \mathsf{Set}$, $f\colon S \to S$, $s\colon S$. Output data: $p\colon \mathbf{N}$, $t\colon \mathbf{N}$ such that $p > 0$, $f^k(s) = f^{k+p}(s)$ for all $k \ge t$ and p and t are the smallest numbers with this property. The input data is such that p and t always exist. Running time: $O(p+t)$. The only allowed operations on elements of S are (1) compute $f(a)$ for some $a\colon S$; (2) check whether $a = b$ for some $a\colon S$, $b\colon S$. Notice that the element s is given to you so that you have something to apply f to.

3. Input data: $S\colon \mathsf{OrderAb}$ (ordered abelian group, e.g., $\mathbf{Z}$, $\mathbf{Q}$, $\mathbf{R}$, etc.; the available operations are abelian group operations and comparison), $m, n\colon \mathbf{N}$, $x\colon S[m]_{\le}$, $y\colon S[n]_{\le}$, $s\colon S$. Output data: $a, b\colon \mathbf{N}$ such that $x[a] + y[b] \ge s$ and $x[a] + y[b] - s$ is as small as possible (with respect to the given order on S). Running time: $O(m + n)$.

4. Input data: $n\colon \mathbf{N}$, $x\colon \mathbf{N}[n]$, x is a permutation of $[0, n)$, i.e., for any $j \in [0, n)$ there is exactly one $i \in [0, n)$ such that $x[i] = j$. Output data: x is transformed into another permutation y (in place) such that $y[x[i]] = i$ and $x[y[i]] = i$ for all $i \in [0, n)$, i.e., y is the inverse of x. Additional memory: $O(1)$ (i.e., y must be computed in place of x, without any new arrays). Running time: $O(n)$.

5. Input data: $S\colon \mathsf{Order}$, $n\colon \mathbf{N}$, $x\colon S[n]_{\le}$. Output data: $m\colon \mathbf{N}$, $y\colon \mathbf{N}[m]_{<}$ (strictly increasing), $y[i] \in [0, n)$ for all $i \in [0, m)$, $x[y[i]] < x[y[j]]$ if $i < j$, $i, j \in [0, m)$, and m is the largest number with these properties. In other words, throw away as few elements of x as possible so that the remaining array is strictly increasing. Running time $O(n \log n)$.

6. Abstract persistent state: $S\colon \mathsf{Set}$, $*\colon S$ (a special element of S), $x\colon S[\mathbf{N}]$ (an abstract infinite array indexed by natural numbers). Initial state: $x[i] = *$ for all $i \in \mathbf{N}$. Operations:

- read($i\colon \mathbf{N}$): returns $x[i]$. Running-time (worst-case or randomized average, your choice): $O(\log n)$, $n = \#\{i \in \mathbf{N} \mid x[i] \ne *\}$.
- insert($i\colon \mathbf{N}, s\colon S$): first, $x[k + 1] \leftarrow x[k]$ for all $k \in [i, j)$ in decreasing order of k, where $j = \min_{l \ge i, x[l]=*}$; then $x[i] \leftarrow s$. Running time: same as above.

Explanation: $*$ means "empty"; when inserting an element s in position i, we first move the entire block starting at i and ending before the first empty position one position to the right, so that $x[i]$ is now empty and can be changed to s.

7. Recall the algorithm that turns a given array $x\colon S[n]$ into a binary heap (meaning $x[\lceil i/2\rceil - 1] \le x[i]$ for all $i \in (0, n)$), namely, for all $j \in [0, n)$ in decreasing order, start with $k \leftarrow j$ and while $x[k]$ is greater than $x[k']$, k' being the index of the smaller son (if it exists), exchange $x[k]$ with $x[k']$ and $k \leftarrow k'$. In the lecture, we proved that this algorithm runs in $O(n)$ time. Compute the average number of comparisons done by this algorithm if x is a random permutation (meaning each permutation occurs with equal probability). For simplicity, assume that $n = 2^a - 1$ for some $a\colon \mathbf{N}$, so that the tree is "full".