Mathematics 5365 (Analysis of Algorithms)

Midterm 2

You may use any algorithm from the lectures without reimplementing it, provided that you cite it correctly (i.e., state what it does exactly, and cite the correct running time). There is no guarantee that any problem can benefit from these algorithms, though.

Solve as many problems as you can. Choose problems that are easiest to solve for you.

1. Input data: $S\colon \mathsf{Order}$, $n\colon \mathbf{N}$, $x\colon S[n]$. Output data: $i, j\colon \mathbf{N}$ such that $i, j \in [0, n)$, $i \neq j$, $x[i] \leq x[k]$ for any $k \in [0, n)$, $x[j] \leq x[k]$ for any $k \in [0, n) \setminus \{i\}$. Thus, $x[i]$ is the smallest element of x and $x[j]$ is the second-smallest element of x. Running time $O(n)$. The total number of comparisons of elements of x must be $n + O(\log n)$, i.e., for all $n \geq n_0$ we must have at most $n + C \log n$ comparisons for some real C and $n_0\colon \mathbf{N}$. Thus, you must find the smallest and the second-smallest element of x using only $n + O(\log n)$ comparisons, and no other operations on elements of x are permitted, i.e., you know nothing about its internal structure. You may assume, if you want, that all elements of x are distinct.

2. Consider the following algorithm. Input data: $S\colon \mathsf{Order}$, $n\colon \mathbf{N}$, $x\colon S[n]$, all elements of x are distinct. Output data: x rearranged in the increasing order. Algorithm:

```
sort(i, j: N)
  if j − i < 2
    return
  k ← split(x[i], i, j)
  sort(i, k)
  sort(k + 1, j)
```

Here $split(a, i, j)$ rearranges the elements of $x[i, j)$ and returns $k \in [i, j]$ such that all elements of $x[i, k)$ are at most a, $x[k] = a$, and all elements of $x(k, j)$ are at least a. In our case, $a = x[i]$, and $split$ requires $j - i - 1$ comparisons, namely, comparing $a = x[i]$ to the elements $x[i + 1], \ldots, x[j - 1]$, of which there are $j - i - 1$.

Compute the average number of comparisons performed by this algorithm. Hint: prove that any pair of distinct elements in x can be compared at most once in the entire algorithm. What is the probability that a given pair of elements will be compared at any point in the algorithm? (You may want to express the answer using a formula that involves the positions of these elements in the final array.) Compute the average number of comparisons using these probabilities.

3. Input data: $n\colon \mathbf{N}$, $a\colon \mathbf{R}[n][3]$. Output data: $m\colon \mathbf{N}$, $p\colon \mathbf{R}[m][2]$, where $p[0], \ldots, p[m - 1]$ are the vertices of the convex polygon $\{(x, y) \in \mathbf{R}^2 \mid \forall i \in [0, n)\colon a[i][0] + a[i][1] \cdot x + a[i][2] \cdot y \geq 0\}$ taken in the counterclockwise order starting from an arbitrary vertex. (Assume, for simplicity, that the intersection is a bounded nonempty subset of $\mathbf{R}^2$.) Running time $O(n \log n)$. In other words, you must intersect n half-planes and compute the vertices of the resulting convex polygon in $O(n \log n)$ time.

4. Input data: $S\colon \mathsf{Set}$, $n\colon \mathbf{N}$, $x\colon S[n]$. Output data: $m\colon \mathbf{N}$, where m is the cardinality of the set $\{\alpha \in S^* \mid \exists i, j \in [0, n)\colon i \leq j \wedge \alpha = x[i, j)\}$. Running time: $O(n^2)$. Extra bonus points (equivalent to one additional problem): running time $O(n)$. Thus, you have to compute the number of distinct substrings of x, where repeating substrings are only counted once. For example, if x = banana, then the above set is $\{\emptyset, \mathrm{a}, \mathrm{b}, \mathrm{n}, \mathrm{ba}, \mathrm{an}, \mathrm{na}, \mathrm{ban}, \mathrm{ana}, \mathrm{nan}, \mathrm{bana}, \mathrm{anan}, \mathrm{nana}, \mathrm{banan}, \mathrm{anana}, \mathrm{banana}\}$, and its cardinality is 16. Another example: $m \in [n + 1, 1 + n(n + 1)/2]$, where $m = n + 1$ corresponds to the case when all elements of x are the same and $m = 1 + n(n + 1)2/$ corresponds to the case when all elements of x are different.

5. Consider the following algorithm. Input data: $p\colon \mathbf{N}$, p is an odd prime, $s\colon \mathbf{F}_p$. Output data: $x\colon \mathbf{F}_p$ such that $x^2 = s$, if such an x exists at all. Algorithm: choose a random element $a \in \mathbf{F}_p^\times$ and terminate if $a^2 - s \notin (\mathbf{F}_p^\times)^2$, otherwise keep choosing a new random element until found. Then compute $x = (a + \sqrt{a^2 - s})^{(p+1)/2}$ in the field $\mathbf{F}_p(\sqrt{a^2 - s}) = \mathbf{F}_p[y]/(y^2 - (a^2 - s))$, i.e., the splitting field of the polynomial $y^2 - (a^2 - s)$. If $x \in \mathbf{F}_p$, then x is the answer, otherwise s has no square root.

- Prove that this algorithm is correct. Make sure to prove that if $x \in \mathbf{F}_p$, then $x^2 = s$ and that if $x \notin \mathbf{F}_p$, then s has no square root. Also make sure to prove (with details of probabilistic arguments) that the algorithm terminates with probability 1.
- How would you implement this algorithm? More precisely: (1) How would you verify that an element of $\mathbf{F}_p$ is not a square? (2) How would you perform the computation of x in the splitting field?
- Compute the average running time. (As usual, you may cite algorithms discussed in the lectures with their running times.)