CS161 Homework 2 **Due:** 22 April 2016, 12 noon
Submit on Gradescope **Handed out:** 15 April 2016

Instructions: Please answer the following questions to the best of your ability. If you are asked to show your work, please include relevant calculations for deriving your answer. If you are asked to explain your answer, give a short ($\sim$ 1 sentence) intuitive description of your answer. If you are asked to prove a result, please write a complete proof at the level of detail and rigor expected in prior CS Theory classes (i.e. 103). When writing proofs, please strive for clarity and brevity (in that order). Cite any sources you reference.

1 (15 points) LexicoSort

In class, we learned how to use CountingSort to efficiently sort sequences of values drawn from a set of constant size. In this problem, we will explore how this method lends itself to lexicographic sorting.

Let $S = `s_1 s_2 ... s_a`$' and $T = `t_1 t_2 ... t_b`$' be strings of length a and b, respectively (we call s_i the ith letter of S). We say that S is *lexicographically less than* T, denoting $S <_{\text{lex}} T$, if either

- $a < b$ and $s_i = t_i$ for all $i = 1, 2, ..., a$, or
- There exists an index $i \leq \min\{a, b\}$ such that $s_j = t_j$ for all $j = 1, 2, ..., i-1$ and $s_i < t_i$.

Lexicographic sorting algorithm aims to sort a given set of n strings into lexicographically ascending order (in case of ties due to identical strings, then in non-descending order).

For each of the following, describe your algorithm clearly, and analyze its running time and prove correctness.

(a) Give an $O(n)$ lexicographic sorting algorithm for n strings where each string consists of exactly one letter from the set a-z (that is, each string is of length 1).

(b) Give an $O(mn)$ lexicographic sorting algorithm for n strings where each string contains letters from the set a-z and is of length at most m. (Hint: pad words of length $< m$.)

(c) Give an $O(mn + mc)$ lexicographic sorting algorithm for n strings of maximum length m with letters drawn from a character set of size c.

2 (15 points) Po Flips Pancakes

Poe the penguin has found a rare ingredient: Kaili yams. He decides to use this special ingredient to make a rare treat: pancakes! After much work, Poe the penguin has made a huge stack of n pancakes on a plate. The pancakes are labelled 1 to n from top to bottom, and pancake i has size P_i ($P_i > 0$ and larger P_i means larger pancake). He decides to feed Barr the Bear with his new creation. Unfortunately, Barr the Bear is an artistic eater and wants the pancake tower to look like a pyramid (i.e. sorted pancakes with largest at the bottom). To appease the bear, Poe decides to use his special pancake tongs to rearrange the pancakes. With the tongs, he can flip the order of i^{th} pancake (from the top) to j^{th} pancake (from the top) for any $1 \leq i \leq j \leq n$. As an example, if the initial pancake sequence of sizes is $[1, 1, 5, 3, 3]$, using the tongs on pancakes 2 to 5 will result in $[1, 3, 3, 5, 1]$ (by flipping $[1, 5, 3, 3]$ into $[3, 3, 5, 1]$). This is a tiring operation however, and will cost Poe $j - i$ units of energy. Poe wants to find a sequence of pairs (i, j) such that flipping the pancakes by following this sequence of pairs will cause the pancakes to become sorted, and so that the chosen flips use little energy.

In this problem we will design algorithms to help Poe. These algorithms will take an array of pancake sizes $\{P_1, \ldots, P_n\}$ as input and will sort the pancakes, assuming that a "flip" takes constant time. (Other

operations such as reading the array etc also take constant time, as usual.) The running time of the algorithm is then computed as usual but assigning $O(1)$ time per flip. In addition to running time, we will analyze the energy of the algorithm. The energy is defined as the sum of the energies of all flips performed by the algorithm.

(a) Suppose all pancakes are of size 1 or 2. Design an $O(n \log n)$ time algorithm that sorts the pancakes using $O(n \log n)$ energy. Analyze the running time and energy of your algorithm and prove correctness.

(Hint: Mergesort.)

(b) Suppose the pancakes are of arbitrary positive sizes. Design an algorithm that runs in $O(n \log^2 n)$ expected time that sorts the pancakes using (expected) $O(n \log^2 n)$ energy. Analyze the running time and energy of your algorithm and prove correctness.

(Hint: Use the previous part to perform quicksort; even if you did not solve the previous part, you may assume that such algorithm exists, and use it as a black-box without justification.)

3 (15 points) Approximate Sorting

We say a sequence of distinct numbers $a_1, a_2, \ldots a_n$ is b-approximately sorted if the following property holds: for each $i = 1, 2, \ldots n$, if a_i is the j-th smallest number in the sequence, then $|i - j| \leq b$. In other words, each number is at most b positions away from its actual sorted index. A b-approximate sorting algorithm is an algorithm that takes a sequence of distinct numbers as input and produces a sequence that is b-approximately sorted.

a) Show that given any n distinct numbers, the number of permutations of those numbers that are $\sqrt{n}$-approximately sorted is $O(n^{cn})$, for some $c < 1$.

b) Use Part (a) to find a lower bound on the number of leaf nodes in the decision tree for any comparison-based $\sqrt{n}$-approximate sorting algorithm, and prove that any such $\sqrt{n}$-approximate sorting algorithm must have worst case complexity $\Omega(n \log n)$.

c) Let $k > 1$ be an arbitrary constant. Can we construct an $\frac{n}{k}$-approximate sorting algorithm that does better than $O(n \log n)$? (Hint: consider the selection algorithm from lecture).

4 (10 points) Testing membership

The goal of this problem is to determine if an element x is a member of a set S.

(a) Describe a deterministic algorithm to determine whether $x \in S$. What is the runtime of this algorithm?

(b) We will walk you through a randomized algorithm for determining whether $x \in S$. In this algorithm, we perform k independent "tests" on x, each test taking $O(1)$ time. The test are relatively reliable, so that:

- Given an element x such that $x \in S$, x always passes the test.
- Given an element x such that $x \notin S$, x fails the test with probability p.

The tests are all independent, so that whether x passes or fails one test does not influence its probability of passing or failing the other tests.

(i) What is the runtime of the randomized algorithm?

(ii) Given that x is not in S, what is the probability that x passes all k tests?

(iii) Let $\epsilon > 0$ be a constant. Suppose that after the k tests we label an element as a member of S if and only if it passes all k tests; otherwise we label it as being not in S. How large should k be so that for any $x \notin S$ the probability that it was incorrectly labeled as being in S, is at most ϵ. (This latter probability is called the "false positive rate".)

(iv) Suppose we are certain that $x \notin S$, and we want to run as many tests as it will take to prove this, even if we have to sit there all day running tests. What is the expected number of tests we would need to run in order to prove that x is not a member of S? What is the variance?