CS 591: Sublinear Algorithms — Professor: Sofya Raskhodnikova
Boston University — September 14, 2020

Homework 2 – Due Thursday, September 17 *before* 10am on Gradescope

Instructions

- Solutions written in LaTeX are strongly preferred, but you can upload any pdf files, including scanned hand-written solutions. Template latex files are on the course webpage.
- Collaboration is allowed and encouraged. However, each of you should think about a problem before discussing it with others and write up your solution independently. You may consult books and online sources to get information about well-known theorems, such as the Chernoff bound. But you are not allowed to look up solutions directly in papers or any other sources. And you *must* list all collaborators and sources! (See full details in the General Information Handout.)
- Correctness, clarity, and succinctness of the solution will determine your score.

Problems

1. This is a collection of questions with short answers (at most several sentences per question).

   (a) Recall that the relative Hamming distance between two strings is the fraction of character positions on which they differ. Give an algorithm for estimating the relative Hamming distance between two strings of the same length within additive error ϵ. Your algorithm should give a good estimate with probability at least 2/3. What's the running time of your algorithm?

   (b) We define the *property* as a set $\mathcal{P}$ of objects (intuitively, the collection of objects that satisfy the property). For example, it can be the set of monotone functions of the form $f : \{0,1\}^d \to \{0,1\}$. Recall that an ϵ-tester for $\mathcal{P}$ has to, with probability at least 2/3,
   - accept objects in $\mathcal{P}$;
   - reject objects that are ϵ-far from $\mathcal{P}$.

   Consider properties $\mathcal{P}_1$ and $\mathcal{P}_2$ such that $\mathcal{P}_1 \subseteq \mathcal{P}_2$. Let $q(\epsilon)$ be some function that represents query complexity. E.g., $q(\epsilon)$ could be $1/\epsilon$ or $1/\epsilon^2$.
   Prove or disprove:
   i. If $\mathcal{P}_1$ has an ϵ-tester that makes $O(q(\epsilon))$ queries then so does $\mathcal{P}_2$.
   ii. If $\mathcal{P}_2$ has an ϵ-tester that makes $O(q(\epsilon))$ queries then so does $\mathcal{P}_1$.
   iii. If $\mathcal{P}_1$ has an ϵ-tester that makes $O(q(\epsilon))$ queries then so does $\overline{\mathcal{P}_1}$.

2. In class we saw an algorithm, based on spanners, for testing if a list of numbers $x_1, \ldots, x_n$ is sorted. Now we will design another algorithm for this problem, based on binary search.

```
BinarySearchSortednessTest($n, \epsilon$)
1  Pick an index $i$ from $\{1, 2, \ldots, n\}$ uniformly at random and read the number $x_i$.
2  Perform a binary search for $x_i$ and **reject** if you find any numbers out of order
   (among all numbers queried, including $x_i$).
```

(a) Analyze the probability that BinarySearchSortednessTest(n, ϵ) rejects a list that is ϵ-far from sorted. **Hint:** Call a number x_i *bad* if it "fails" the binary search, i.e., Step 2 performed on x_i would reject.

(b) Prove that, with enough repetitions, BinarySearchSortednessTest(n, ϵ) is an ϵ-tester for sortedness.

(c) How does the query complexity and running time of this test compare to those of the test we saw in class?

(d) An algorithm is called *nonadaptive* if its queries do not depend on answers to previous queries. Otherwise, it is called *adaptive*. Was the spanner-based algorithm adaptive? Is your new algorithm adaptive? **Hint:** One of them is adaptive and the other is nonadaptive. Can you modify the adaptive algorithm to make it nonadaptive without changing its running time?

3. In class we saw a tester for connectedness that made $O(\frac{1}{\epsilon^2 d})$ queries. Give a tester for connectedness that makes $O(\frac{1}{\epsilon}\text{polylog}\frac{1}{\epsilon^2 d})$ queries, where polylog m means that there is a constant c such that the expression is $\log^c m$.

   Hint: In class we proved that if a graph is ϵ-far from connected, it has many small connected components. ("Many" was $\geq \frac{\epsilon d n}{4}$ and "small" was of size $\leq \frac{4}{\epsilon d}$.) Try to do a more careful accounting by considering small components of different sizes separately. I.e., break components into *buckets* according to their size (1, 2 to 3, 4 to 7, etc.) and prove that at least one of the buckets contains "enough" components. Modify the test accordingly.