HW 7

Due on **Wednesday, October 20, 2021 at 10am**

CS 473: Algorithms, Fall 2021 Version: **1.1**

Submission guidelines and policies as in homework 1.

19 (100 PTS.) Min cut questions.

19.A. (50 PTS.) Consider a graph $G = (V, E)$ with n vertices, m edges and a min cut of size k. Let $\mathcal{F}$ be the collection of all min-cuts in G (i.e., all the cuts in $\mathcal{F}$ are of size k). What is the probability that MinCut (the simpler variant – see class notes) would output a specific min-cut $S \in \mathcal{F}$?

Using this lower bound on the probability, bound the size of $\mathcal{F}$ (hint: if an algorithm outputs each element of a set S with probability $\geq \alpha > 0$, what is a natural upper bound on the size of S?).

19.B. (50 PTS.) A ***connected cut*** is a cut $(S, \overline{S})$ such that the induced subgraphs G_S and $G_{\overline{S}}$ are connected.

Consider a specific connected cut $C = \left(S, \overline{S}\right)$ with kt edges in it, where $t \geq 1$. What is the probability that MinCut would output this cut? (Again, provide a lower bound on this probability.)

Next, consider the set $\mathcal{F}(t)$ of all connected cuts in G of size at most kt. Bound the size of $\mathcal{F}(t)$ using the above.

20 (100 PTS.) Randomized algorithms types.

Consider a randomized algorithm – there are several types of such algorithms, including:

(I) ***Monte Carlo algorithms***: The algorithm might be wrong with low probability, but the running time is polynomial. (For example, the algorithm for min-cut seen in class.)

(II) ***Las Vegas algorithm***: The algorithm always output the correct answer, but the running time is a random variable. QuickSort is one example of such an algorithm.

20.A. (30 PTS.) Consider a Las Vegas randomized algorithm with expected running time $\mu(n)$. For concreteness, say $\mu(n) = \Theta(n^2)$. Prove that the probability the algorithm takes more than $4\mu(n)$ time, is at most $1/4$.

20.B. (30 PTS.) For the algorithm above, and a prescribed $\delta \in (0, 1)$, describe how to modify the algorithm such that its running time is $O(\mu(n) \log(1/\delta))$ with probability $\geq 1 - \delta$.

20.C. (40 PTS.) Assume you are given a Monte-Carlo algorithm for a problem (e.g., computing minimum cut in a graph), that runs in $O(T(n))$ time, where its output is correct with probability $\geq 1/2$. Furthermore, assume that you are given a polynomial time verifier – given a solution, it can tell if it is correct, that runs in $O(V(n))$ time. Present a Las Vegas algorithm for the given problem, with expected running time $O\big(T(n) + V(n)\big)$ – that is an algorithm that its result is always correct.

21 (100 PTS.) A bit on hashing.

21.A. (50 PTS.) Consider an algorithm that stores m elements in an array B of size n – as follows. It randomly choose for each element a random cell in uniform from the cells of B. If two or more elements get stored in the same cell in memory we throw them away. That is, we keep only the elements that get mapped to their own unique cell.

What is the expected number of elements that get thrown away? Let this number be $f(m)$.

21.B. (50 PTS.) Consider playing the above starting with n elements. In the ith iteration, we throw the elements that were not stored yet into a new array $B_i[1 \,..\, n]$, keeping in the array only the elements that got mapped to their own cells (with no other elements mapped to this cell), and moving all the others to the next round. We would like to bound the number of rounds one needs till all the elements are stored. Proving the right bound here is somewhat technically tedious and painful, so we are going to make a simplifying assumptions – the number of elements moving to the next round is exactly the expected number of such elements (this is up to a constant a correct assumption, but proving it requires tools that outside the scope of this class).

Give an upper bound, as tight as possible, on the number of rounds one needs to play, till all the elements are stored (under the above assumption). **Prove** your answer. (Your upper bound should a clean bound – not a complicated formula.)

Getting the right answer here is hard. It is not too difficult to get a suboptimal but still somewhat interesting answer, which would be worth half the points of the question.