**EECS 376: Foundations of Computer Science** **Due 8:00pm, April 10**
**University of Michigan, Winter 2024** **Homework 10** **(accepted until 9:59 pm, no credit after)**

This homework has 8 questions, for a total of 100 points and 5 extra-credit points.
Unless otherwise stated, each question requires *clear*, *logically correct*, and *sufficient* justification to convince the reader.
For bonus/extra-credit questions, we will provide very limited guidance in office hours and on Piazza, and we do not guarantee anything about the difficulty of these questions.
We strongly encourage you to typeset your solutions in LaTeX.
If you collaborated with someone, you must state their name(s). You must *write your own solution* for all problems and *may not use any other student's write-up*.

(0 pts) 0. **Before you start; before you submit.**

If applicable, state the name(s) and uniqname(s) of your collaborator(s).

**Solution:**

(10 pts) 1. **Self assessment.**

Carefully read and understand the posted solutions to the previous homework. Identify one part for which your own solution has the most room for improvement (e.g., has unsound reasoning, doesn't show what was required, could be significantly clearer or better organized, etc.). Copy or screenshot this solution, then in a few sentences, explain what was deficient and how it could be fixed.

(Alternatively, if you think one of your solutions is significantly *better* than the posted one, copy it here and explain why you think it is better.)

If you didn't turn in the previous homework, then (1) state that you didn't turn it in, and (2) pick a problem that you think is particularly challenging from the previous homework, and explain the answer in your own words. You may reference the answer key, but your answer should be in your own words.

**Solution:**

2. **Job scheduling.**

Consider the problem of *fastest-completion job scheduling*: allocating a collection of jobs to servers so as to minimize the time until all the jobs are completed. An input is a number of identical servers $n$, and the times $t_1, \ldots, t_m$ that the $m$ jobs require, i.e., $t_i$ is how long it takes any server to complete job $i$. The desired output is an allocation of the jobs to the servers having minimum *completion time*, where the completion time of an allocation is the maximum, over all the servers, of the total time of the server's allocated jobs. Each job must be allocated to a single server.

It turns out that this is an NP-complete problem, but there is a polynomial-time (greedy) 2-approximation algorithm:

```
1: function GreedyAllocate(n, t_1, ..., t_m)
2:     while there is an unallocated job do
3:         allocate an arbitrary unallocated job to a server that would finish its already-allocated jobs earliest
4:     return the computed allocation
```

(6 pts) (a) Suppose that there are $n = 3$ servers and $m = 7$ jobs, where $t_1 = \cdots = t_6 = 1$, and $t_7 = 3$. Determine, with justification, an optimal allocation and its completion time.

Then, consider running GreedyAllocate on this input. Show that, depending on the order in which jobs are allocated by the algorithm, the resulting allocations can have different completion times and approximation factors. Specifically, give two sequences of choices made by the algorithm that result in different completion times, and for each ordering, give its approximation ratio relative to the optimal completion time.

**Solution:**

(6 pts) (b) Prove that when $m \leq n$, GreedyAllocate necessarily returns an optimal allocation.

**Solution:**

(6 pts) (c) We have handled the case $m \leq n$, so suppose that $m > n$. Letting OPT be the optimal completion time, prove that $\text{OPT} \geq \max\{t_1, t_2, \ldots, t_m\}$, and $\text{OPT} \geq \frac{1}{n} \cdot \sum_{j=1}^{m} t_j$.

**Solution:**

(6 pts) (d) Prove that $\text{ALG} \leq 2 \cdot \text{OPT}$, where ALG is the completion time obtained by GreedyAllocate.

*Hint*: Consider a server that finishes last (i.e., it finishes at the completion time), and how the jobs were allocated before this server's final job was allocated to it.

**Solution:**

3. **Indicator variables and linearity of expectation.**

Let $G_{n,p}$ denote a random undirected graph on $n$ vertices, constructed as follows: for each (unordered) pair of distinct vertices $u, v \in V$, include the edge $(u, v)$ in $G_{n,p}$ with probability $p$, independently of all other random choices.

(6 pts) (a) Derive the expected number of edges in $G_{n,p}$.

**Solution:**

(6 pts) (b) Derive the expected degree of any vertex in $G_{n,p}$.

**Solution:**

(6 pts) (c) A *triangle* in a graph is a set of three (distinct) vertices that have an edge between every pair of them. Derive the expected number of triangles in $G_{n,p}$.

**Solution:**

(6 pts) (d) Prove that the number of triangles in $G_{n,p}$ is at least $n^3p^2$ with probability at most $p/6$.

**Solution:**

4. **Randomized max-cut.**

In this problem, all graphs are undirected and (as usual) have *no self-loops*, i.e., there is no edge from a vertex to itself. For a cut $S \subseteq V$ in a graph $G = (V, E)$, let $C(S) \subseteq E$ denote the subset of edges "crossing" the cut, i.e., those that have exactly one endpoint in $S$. The size of the cut is then $|C(S)|$. Consider the following randomized algorithm that outputs a cut in a given graph $G = (V, E)$.

```
initialize S = ∅
for all v ∈ V do
    put v into S with probability 1/2, independently of all others
return S
```

(7 pts) (a) Define suitable indicator variables and use linearity of expectation to prove that *in expectation*, the above algorithm obtains a 1/2-approximation for MaxCut. That is, the expected size of the output cut is at least half the size of a maximum cut.

**Solution:**

(7 pts) (b) Prove that $\Pr[|C(S)| \geq (1-\varepsilon)|E|/2] \geq \frac{\varepsilon}{1+\varepsilon}$ for any $\varepsilon > 0$.
Notice that for $\varepsilon = 1/(2|E|)$, this is a lower bound on the probability that the number of edges crossing $S$ is at least $|E|/2$, because the number of crossing edges is an integer.

**Solution:**

(5 pts) (c) As a stepping stone to the next part, we consider the following intermediate question.
Consider a probability experiment that consists of a sequence of "attempts," where each attempt succeeds with probability $p$, independently of all others. We keep making attempts until one succeeds, at which point the experiment terminates.
Let $X$ denote the number of attempts until termination (including the attempt that finally succeeds). Prove that

$$\mathbb{E}[X] = p + (1-p)(1 + \mathbb{E}[X]) \ ,$$

and conclude that $\mathbb{E}[X] = 1/p$. (The distribution of $X$ is known as the *geometric distribution* with success probability $p$.)

**Solution:**

(6 pts) (d) Suppose we repeatedly run our randomized MaxCut algorithm until we get a cut of size at least $|E|/2$. Derive an upper bound on the expected number of attempts that are needed.

**Solution:**

(5 pts) 5. **Quicksort.**

In class, we showed that (randomized) Quicksort has expected running time $O(n \log n)$ on an array of $n$ elements. Although the worst-case running time of the algorithm is $\Omega(n^2)$, we will prove that this happens with small probability.

Let $c_1$ be the constant such that the expected running time of Quicksort is at most $c_1 \cdot n \log n$ (for all large enough $n$). Prove that, for any constant $c_2 > 0$, the probability that it takes time at least $c_2 n^2$ time is $O(\frac{\log n}{n})$.

**Solution:**

(12 pts) 6. **Random binary search trees.**

A binary search tree on distinct values is a binary tree where for any node with value $x$, all the nodes in its left subtree have values less than $x$, and all nodes in its right subtree have values greater than $x$. Consider the random process that generates a binary search tree on the integers $\ell, \ldots, r$ for given $\ell \leq r$, as follows.

- Choose $x \in \{\ell, \ldots, r\}$ uniformly at random and take $x$ as the root. If $\ell = r$, output the tree consisting of just $x$.
- If $\ell < x$, for $x$'s left subtree, recursively build a random binary search tree on $\ell, \ldots, x-1$.
- If $x < r$, for $x$'s right subtree, recursively build a random binary search tree on $x+1, \ldots, r$.

For a random binary search tree on $1, \ldots, n$, prove that the expected depth of the node with any particular value is $O(\log n)$.

*Hint*: Use ideas from the analysis of the Quicksort algorithm from class.

**Solution:**

(5 EC pts) 7. **Optional extra-credit question: randomized vertex-cover.**

This question considers randomized algorithms for approximating the vertex-cover problem.

(a) Consider the following probability experiment:

- There is a fixed set of $k$ distinct items.
- The experiment proceeds in a sequence of rounds. In each round, with probability at least $1/2$ (and independently of all other rounds), some arbitrary item that was not previously chosen is chosen. (Otherwise, nothing is chosen in this round.)
- Once every item has been chosen, the experiment ends.

Let $R$ be the random variable denoting the number of rounds until the experiment ends. Prove that $\mathbb{E}[R] \leq 2k$.

*Hint:* for $j = 1, \ldots, k$, let $R_j$ be the random variable denoting the number of rounds after the $(j-1)$st chosen item and until the $j$th chosen item (inclusive). What is the relationship between $R$ and the $R_j$? What is an upper bound on $\mathbb{E}[R_j]$?

**Solution:**

(b) Prove that the following randomized algorithm outputs a 2-approximation, in expectation, for the minimum vertex cover problem.

*Hint:* let $C^* = \{v_1, \ldots, v_k\}$ be some minimum vertex cover in the input graph. Show how a run of the algorithm corresponds to a (complete or partial) run of the experiment from the previous part with the $v_i$ as the items, and use this to bound the expected size of the output $C$.

```
1: function RandomSingleCover(G = (V, E))
2:     C = ∅
3:     while there is some edge e = (u, v) that is not covered by C do
4:         add exactly one of u or v to C, each with probability 1/2
5:     return C
```

**Solution:**