Homework 5. Due March 13

CS180: Algorithms and Complexity
Winter 2017

GUIDELINES:

- Upload your assignments to Gradescope by 6:59 PM.
- Follow the instructions mentioned on the course webpage for uploading to Gradescope very carefully (including starting each problem on a new page and matching the pages with the assignments); this makes it easy and smooth for everyone. As the guidelines are simple enough, bad uploads will not be graded.
- You may use results or algorithms proved in class without proofs as long as you state them clearly.
- Most importantly, make sure you adhere to the policies for academic honesty set out on the course webpage. The policies will be enforced strictly. Homework is a stepping stone for exams; keep in mind that reasonable partial credit will be awarded and trying the problems will help you a lot for the exams.

1. Consider the complete rooted ternary tree of depth n. That is take a rooted tree where the root has three children, each child has exactly 3 children and so on for n levels. So for example for $n = 1$, you have the root connected to its three children (with 4 nodes in total); for $n = 2$ you have 13 nodes in total; and more generally, the total number of nodes in the tree of depth n is exactly $1 + 3 + 3^2 + \cdots 3^n = (3^{n+1} - 1)/2$.

   Now, suppose that you delete each edge of the tree independently with probability $1/2$. Let X be the number of nodes which are still connected to the root after the deletion of the edges. Compute the expectation of X (with proof). [.75 points]

2. Call a sequence of coin tosses "monotone" if the sequence never changes from Heads to Tails when parsed left to right. For example, the sequences $TTTHHHH$, $TTTT$ are monotone whereas $TTTHHHT$ is not. Consider a sequence of n coin tosses of a fair coin. For integer $k > 0$, let Y_k be the number of monotone sub-sequences of length k in the n coin tosses. Compute the expectation of Y_k (with proof). [.4 points]

   Recall that a sequence x is a sub-sequence of y if x can be obtained from y by deleting some symbols of y without changing the order of the remaining elements.

3. Define a random graph $G = (V, E)$ as follows. The vertex set of G is $V = \{1, \ldots, n\}$. Now for each pair $\{i, j\}$ with $i \neq j \in [n]$, add the pair $\{i, j\}$ to E with probability $1/2$ independent

of the choice for every other pair. Let X_5 be the number of independent sets of size 5 in the graph G. Compute the expectation of X_5 (with proof). [.35 points]

Recall that an independent set is a collection of vertices I in G such that no two vertices in I have an edge between them.

Hint For problems 1,2,3, one way is to write the the main random variable as a sum of 'simpler' random variables and use linearity of expectation to compute the expectation.

4 Let A be an array of n positive integers and $W = (w_1, \ldots, w_n)$ be a list of positive integers where we view w_i as the *weight* of a_i. For an integer $k \geq 1$, let $\textsc{WSelect}(A, k)$ be the largest element a in A such that the total weight of items not larger than a in A is at most k. That is, $\textsc{WSelect}(A, k)$ is defined to be the largest element a in A such that $\sum_{i:a_i \leq a} w_i \leq k$. If there is no such element in A, then we set $\textsc{WSelect}(A, k) = 0$.

For instance, if $w_i = 1$ for all the items $i = 1, \ldots, n$, then $\textsc{WSelect}(A, k)$ is just the k'th smallest element of A.

Give a randomized algorithm which given as input a list of n positive integers A, the associated weights W, and an integer $k \geq 1$ finds $\textsc{WSelect}(A, k)$. You can assume that the elements of A are distinct. To get full-credit your algorithm should always output the correct answer and must run in expected $O(n)$ time. You do not need to prove correctness or analyze the time-complexity of the algorithm. [.75 points]

5* Suppose you are given n apples and n oranges. The apples are all of different weights and all the oranges have different weight. However, for each apple there is a corresponding orange of the same weight and vice versa. You are also given a weighing machine that (counter to common intuition) will **only** compare apples and oranges: that is, if you feed an apple and an orange to the machine it will tell you whether the apple or the orange is heavier or if they have the same weight. Your goal is to use this machine to pair up each apple with the orange of the same weight with as few uses of the machine as possible.

Give a randomized algorithm to determining the pairing. For full-credit, your algorithm should only use the machine $O(n \log n)$ times in expectation. You do not need to prove correctness or analyze the time-complexity of the algorithm. [.75 points]

(Remember that you cannot compare an apple to an apple or an orange to an orange.)