# CS:3330 Homework 5, Spring 2017

## Due at the start of class on Thu, March 2

Since you have less than a week to complete this homework, the number of problems in this homework are fewer than usual and all three problems are familiar to you.

1. Consider the "Shortest Interval first" greedy algorithm for the *Interval Scheduling* problem. In this algorithm, we repeatedly pick a shortest interval to include in our solution and as usual when an interval $I$ is picked, then $I$ and any overlapping intervals still present are deleted. The algorithm breaks ties arbitrarily; in other words, if there are multiple shortest intervals present, the algorithms picks one arbitrarily.

   (a) Show that this algorithm does *not* solve the Interval Scheduling problem. In other words, even though the algorithm returns a non-overlapping set of intervals, the set it returns need not be the largest possible set. (We have discussed a counterexample for this algorithm in class.)

   (b) Let $A$ be the set of intervals returned by the algorithm for some input and let $O$ be an optimal solution for this input. Prove that every interval in $A$ overlaps at most two intervals in $O$.
   **Note:** This is not a long proof, but requires care.

   (c) Consider an arbitrary input and let $A$ be the set of intervals returned by the algorithm for this input and let $O$ be an optimal solution for this input. Now for each interval $x$ in $O$, charge \$1 to an interval $y$ in $A$ that overlaps $x$. Note that $y$ could be identical to $x$. Also, note that $y$ has to exist; otherwise the greedy algorithm would have added $x$ to the set $A$. Thus the number of dollars charged is exactly equal to $|O|$. Now answer the following questions: (i) what is the maximum number of dollars that an interval in $A$ is charged? (ii) what does this tell us about the relative sizes of $A$ and $O$? (Express your answer as an inequality connecting $|A|$ and $|O|$.), and (iii) what does this tell us about the "shortest interval first" algorithm being an approximation algorithm for Interval Scheduling?

2. The *Bin Packing* problem takes as input an infinite supply of *bins* $B_1, B_2, B_3, \ldots$, each bin of size 1 unit. We are also given $n$ *items* $a_1, a_2, \ldots, a_n$ and each item $a_j$ has a size $s_j$ that is a real number in the interval $[0, 1]$. The Bin Packing problem seeks to find the smallest number of bins such that all $n$ items can be packed into these bins.

   For example, suppose that we are given 4 items $a_1, a_2, a_3$ and $a_4$ of sizes 0.5, 0.4, 0.6, and 0.5 respectively. We could pack $a_1$ and $a_2$ in bin $B_1$ because $s_1 + s_2 = 0.9 \leq 1$. We could then pack $a_3$ into bin $B_2$, but we could not also add $a_4$ to bin $B_2$, because $s_3 + s_4 = 1.1 > 1$. So $a_4$ would have to be packed in bin $B_3$. This gives us a bin packing of the 4 items into three bins. An alternate way of packing items that would lead to the use of just two bins is to pack $a_1$ and $a_4$ into bin $B_1$ and $a_2$ and $a_3$ into bin $B_2$. This packing that uses only two bins is an optimal solution to the Bin Packing problem.

   The *First Fit* greedy algorithm processes items in the given order $a_1, a_2, \ldots, a_n$ and it considers the bins in the order $B_1, B_2, \ldots$. For each item $a_j$ being processed, the algorithm packs $a_j$ into the first bin that has space for it. It turns out that this very simple algorithm is a 2-approximation algorithm for Bin Packing. The following problems will help you prove this.

   (a) Suppose that the First Fit algorithm packs the given items into $t$ bins. Prove that at most one of these bins is more than half-empty. Use this to deduce that the total size of the $n$ input items is at least $(t-1)/2$.

   (b) Use what you showed in (a) to then show that if an optimal bin packing uses $b^*$ bins, then the First Fit algorithm uses at most $2b^* + 1$ bins.

3. We are given a length-$n$ binary list $L$. In other words, $L[j] \in \{0, 1\}$ for all $j = 1, 2, \ldots, n$. Our problem is to count the number of 0's in $L$. We could of course solve this problem in $\Theta(n)$ time by simply scanning $L$, but we want to solve it faster and so we turn to randomization. The randomized algorithm we use is the following.

```
function CountZeroes(L)
    n ← length(L)
    count ← 0

    Comment: Here C is a positive integer constant
    for j ← 1 to C do
        Comment: pick a random index i between 1 and n
        i ← random(1, n)
        if L[i] = 0 then
            count ← count + 1

    return n · count/C
```

A random variable $X$ has *binomial distribution* with parameters $m$ (a positive integer) and $p$ (a real number between 0 and 1) if

$$\Pr[X = k] = \binom{m}{k} \cdot p^k \cdot (1-p)^{m-k}. \tag{1}$$

The way to think about $X$ being binomially distributed is that we perform $m$ independent random trials and each trial can either succeed or fail. The "success" probability of a trial is $p$ and so the failure probability of a trial is $1 - p$. Then the random variable $X$ represents the number of successful trials, out of $m$ total random trials. To understand the formula in (1) note that the probability of $k$ successful trials is $p^k$, the probability of $(m - k)$ unsuccessful trials is $(1 - p)^{m-k}$, and there are $\binom{m}{k}$ ways of distributing the $k$ successful trials among the $m$ random trials. A fact that is well known about the binomial distribution is that a random variable $X$ that has binomial distribution with parameters $m$ and $p$ has expectation $m \cdot p$.

(a) Suppose that $n = 10^6$ and exactly a quarter of the elements in $L$ are 0, i.e., the number of 0's in $L$ is $10^6/4$. What is the expected answer returned by CountZeroes?

(b) As you know, we would like CountZeroes to return a number close to $10^6/4$. Specifically, suppose that we want CountZeroes to return an answer that is within 10% of the correct answer, i.e., in the range $[10^6/4 - 10^5/5, 10^6/4 + 10^5/4]$. Intuitively it seems that as $C$ (which is the number of times we sample from $L$) increases it becomes more likely that CountZeroes will produce an answer in this range. This problem asks that you calculate the probability that CountZeroes returns a answer in the range $[10^6/4 - 10^5/5, 10^6/4 + 10^5/4]$, as a function of $C$. Since it seems a bit difficult to find a exact, closed-form formula for this probability, we want you to implement a small function that takes as input a positive integer $C$ and returns the probability that CountZeroes returns an answer in the range $[10^6/4 - 10^5/5, 10^6/4 + 10^5/4]$. Use this function to compute this probability for $C = 100, 200, 300, \ldots, 1000$. Show your answers in a table and discuss (in 1-2 sentences) whether your results support our intuition that as $C$ increases the probability we're interested in also increases.