Sample Solutions to Problem Set 6

(The solutions to the exercises from the book are taken from those provided by the authors.)

1. (12 points) Network flows and atmospheric science experiments

(a) Define a flow network as follows. There is a source s, a node x_i representing each balloon i, a node z_i representing each condition c_i, and a sink t. There are edges (s,x_i) of capacity 2, (x_i,z_j) of capacity 1 whenever $c_j \in S_i$, and edges (z_j,t) of capacity k. We then test whether the maximum s-t flow has value nk.

The Ford-Fulkerson algorithm to find a maximum flow has running time $O(|E|C)$, where $|E|$ is the number of edges and C is the total capacity of edges out of s. Here we have $|E| = O(mn)$ and C=$2m$, so the running time is $O(m^2n)$.B

(b) Break all the edges (x_i,z_j). Insert new nodes y_pj for each sub-contractor p and condition c_j. Add an edge (x_i,y_{pj}) of capacity 1 where $c_j \in S_i$ and balloon i is produced by sub-contractor p. Add an edge (y_{pj},c_j) of capacity k-1. Again test whether the maximum s-t flow has value nk.

As in part (a), the running time is $O(|E|C)$. Here $|E|$ is still $O(mn)$, and C=$2m$, so the running time is $O(m^2n)$.

2. ($4 \times 4 = 16$ points) Average-case analysis of insertion sort

Insertion sort is a simple iterative sorting algorithm, the pseudocode for which is given below.

```
InsertionSort(A)
1. for j ← 2 to n
2.     key = A[j]
3.     i ← j − 1
4.     while i > 0 and A[i] > key
5.             A[i + 1] ← A[i]
6.             i ← i − 1
7.     A[i + 1] ← key
```

(a) Prove that the worst-case running time for insertion sort is $\Theta(n^2)$.

Answer: The worst case running time for insertion sort is $\Theta(n^2)$: the cost of each iteration of the inner *while* loop is bounded from above by $O(1)$ (contant) time, the indices i and j are both at most n, and the inner loop is executed once for each of the n^2 pairs of values for i and j in the worst case.

While the worst-case running time of both insertion sort and quicksort is $\Theta(n^2)$, quicksort is considered an efficient sorting algorithm because the randomized version of quicksort has an expected

running time of $\Theta(n \lg n)$ (as we have seen in class). We now consider how insertion sort performs on average.

Let $A[1..n]$ be an array of distinct numbers. If $i < j$ and $A[i] > A[j]$, then we say that the pair (i, j) is an *inversion* of A.

(b) When does an array have the minimum number of inversions? What is this minimum number? When does an array have the maximum number of inversions? What is this maximum number?

Answer: A fully sorted array has 0 inversions since whenever $i < j$, $A[i] \leq A[j]$. Therefore, a fully sorted array has the minimum number of inversions.

On the other hand, the array that is inverse sorted has the maximum number of inversions since every pair (i, j) (for $i < j$) is an inversion. How many pairs (i, j) such that $i < j$ are there? For $i = 1$, there are $n - 1$ possible values for j. For $i = 2$, there are $n - 2$ possible values for j. In general, for i, there are $n - i$ possible values for j. Therefore, the totol number of inversions in an inverse sorted array equals:

$$\sum_{i=1}^{n}(n - i) = \sum_{k=0}^{n-1} k = \frac{(k-1)k}{2}.$$

(c) If $T(A)$ is the running time of insertion sort on input A and $I(A)$ is the number of inversions in A, then prove that $T(A)$ is $\Theta(I(A) + n)$.

Answer: Consider what happens in the jth for loop of insertion sort. At the start of this for loop, all of the elements in $A[1..j-1]$ have been sorted. In the for loop, we first set k to $A[j]$ (line 2) and then compare it with elements $A[j-1]$, $A[j-2]$, ..., (the while-loop in lines 5 through 7) until an element smaller than or equal to k is found. Thus the total number of executions of the while-statement is exactly equal to 1 more than the number of elements that are in the first $(j-1)$ positions and are greater than $A[j]$. But the elements that are in the first $(j-1)$ positions and are greater than $A[j]$ exactly correspond to the number of inversions of the form $(\cdot, j)$ in A. Thus, if we add over all of the for loops, we get that the total number of while loop executions is $I(A) + (n-1)$. Since there are only a constant number of operations in each while loop, the running time of insertion sort is $\Theta(I(A) + n)$.

(d) Show that if A is a random permutation chosen uniformly at random from the set of all $n!$ permutations, then the expected number of inversions is $\Theta(n^2)$. Now invoking part (c), derive a tight bound on the expected running time of insertion sort on a random permutation.

Answer: Consider a pair (i, j). If we select a random permutation, what is the probability that this pair is an inversion? Since both $A[i] > A[j]$ and $A[j] > A[i]$ are equally likely, this probability is $1/2$. Therefore, any pair (i, j) contributes 1 to the total number of inversions with probability $1/2$, and 0 with probability $1/2$. Therefore, the expected contribution of each pair is $1/2$.

As we have seen, there are a total of $n(n-1)/2$ possible pairs (i, j). Since the expected contribution of each pair is $1/2$, the expected total number of inversions $I(A)$ is $n(n-1)/4$. Therefore, by part (c), the expected running time equals $\Theta(n(n-1)/4 + n) = \Theta(n^2)$.

3. (12 points) Random generation of peer-to-peer networks

(a) For every node v_k that comes later than v_j, i.e. $k \text{ ¿ } j$, it has probability $\frac{1}{k-1}$ to link to v_j, since v_k chooses from the k-1 existing nodes with equal probabilities. For all the nodes coming before v_j, such probability is obviosuly zero.

So the expected number of incoming links to node v_j is

$$\begin{aligned}\sum_{k=j+1}^{n} \frac{1}{k-1} &= \sum_{k=1}^{n-1} \frac{1}{k} - \sum_{k=1}^{j-1} \frac{1}{k} \\ &= H(n-1) - H(k-1) \\ &= \Theta(\ln n) - \Theta(\ln k) \\ &= \Theta(\ln \frac{n}{k})\end{aligned}$$

(In the above equations, $H(m)$ refers the mth Harmonic number.)

(b) Consider a node v_j, every node v_k with $k > j$ has probability $1 - \frac{1}{k-1}$ not to link to v_j.

Define the random variable X_j.

$$X_j = \begin{cases} 1 & \text{if node } v_j \text{ has no incoming links and} \\ 0 & \text{otherwise} \end{cases}$$

We obtain

$$\begin{aligned}E[X_j] &= \Pr[\text{no nodes links to } v_j] \\ &= \prod_{k=j+1}^{n} (1 - \frac{1}{k-1}) \\ &= \frac{j-1}{j}\frac{j}{j+1}\frac{j+1}{j+2}\frac{n-2}{n-1} \\ &= \frac{j-1}{n-1}\end{aligned}$$

Therefore, by linearity of expectations, we get the expected number of nodes without incoming links

$$\begin{aligned}\sum_{j=1}^{n} E[X_j] &= \sum_{j=1}^{n} \frac{j-1}{n-1} \\ &= \frac{1}{n-1}\frac{n(n-1)}{2} \\ &= \frac{n}{2}.\end{aligned}$$

4. (12 points) One-pass auctions

The strategy is as follows. The seller watches the first $n/2$ bids without accepting any of them. Let b' be the highest bid among these. Then, in the final $n/2$ bids, the seller accepts any bid that is larger than b'. (If there is no such bid, the seller simply accepts the final bid.)

Let b_i denote the highest bid, and b_j denote the second highest bid. Let S denote the underlying sample space, consisting of all permutations of the bids (since they can arrive in any order.) So $|S|=n!$. Let E denote the event that b_j occurs among the first $n/2$ bids, and b_i occurs among the final $n/2$ bids.

What is $|E|$? We can place b_j anywhere among the first $n/2$ bids ($n/2$ choices); then we can place b_i anywhere among the final $n/2$ bids ($n/2$ choices); and then we can order the remaining bids arbitrarily ($(n\text{-}2)!$ choices). Thus $|E|=\frac{1}{4}n^2(n\text{-}2)!$, and so

$$\Pr[E] = \frac{n^2(n-2)!}{4n!} = \frac{n}{4(n-1)} \geq \frac{1}{4}$$

Finally, if event E happens, then the strategy will accept the highest bid, so the highest bid is accepted with probability at least 1/4.

Bonus Problem. (6 points) A fast network flow algorithm

The statement is false and following is the counterexample to it. Let us be given a number $b > 1$ (we will without loss of generality assume that it is an integer, otherwise we will round it up). We consider the following graph. It has $2(b+1)+2$ vertices: source s, sink t, and vertices u_1, u_2,...,u_{b+1} that have an edge coming from the source and vertices v_1,v_2,...,v_{b+1} that have an edge going into the sink. There is also an edge from u_i to v_i and from v_i to u_{i+1}. All the edge capacities are 1.

Now assume that the first augmenting path was the path $s \to u_1 \to v_1 \to u_2 \to v_2 \to ... \ u_{b+1} \to v_{b+1} \to t$. Then since all the backward edges are deleted from the residual graph according to the super-fast algorithm, the residual graph would contain no path from s to t, and therefore our final flow would equal 1. But there is a flow of value b+1 by using the horizontal edges (that is $u_i \to v_i$). Therefore we failed to reach within b of the optimum.

Notice that for different b's we would be considering different graphs, but we are allowed to do this, since the problem asks whether there exists a *universal* b that is independent of the choice of the flow graph G.