Homework 2

Aleck Zhao

September 20, 2017

1 Reversals

Given an array $[a_1, a_2, \cdots, a_n]$, a reversal is a pair (i, j) such that $i < j$ but $a_i > a_j$. For example, in the array $[5, 3, 2, 10]$ there are three reversals $\{(1, 2), (1, 3), (2, 3)\}$. Note that the array has no reversals if and only if it is sorted, so the number of reversals can be thought of as a measure of how well-sorted an array is.

(a) What is the expected number of reversals in a random array? More formally consider a random permutation of n distinct elements $a_1, \cdots, a_n$: what is the expected number of reversals?

Solution. WLOG, let the array be a permutation of $[1, 2, \cdots, n]$. Let I_{ij} be the indicator defined as

$$I_{ij} = \begin{cases} 1 & \text{if } a_i > a_j \\ 0 & \text{if } a_i \leq a_j \end{cases} \quad i < j$$

That is, $I_{ij} = 1$ if and only if (i, j) is a reversal. The number of reversals R is the sum of all indicators:

$$R = \sum_{i=1}^{n} \sum_{j=i+1}^{n} I_{ij}$$

$$\implies E[R] = E\left[\sum_{i=1}^{n} \sum_{j=i+1}^{n} I_{ij}\right] = \sum_{i=1}^{n} \sum_{j=i+1}^{n} E[I_{ij}]$$

Now, since I_{ij} is an indicator variable, we have $E[I_{ij}] = P(a_i > a_j)$. Now, by the law of total probability,

$$P(a_i > a_j) = \sum_{k=1}^{n} P(a_i > a_j \mid a_j = k) P(a_j = k)$$

Since each k is equally likely, $P(a_j = k) = \frac{1}{n}$, and there are $n - k$ elements greater than k for a_i, all equally likely out of $n - 1$ remaining values. Thus, this is equal to

$$P(a_i > a_j) = \sum_{k=1}^{n} \frac{n-k}{n-1} \cdot \frac{1}{n} = \frac{1}{n(n-1)} \sum_{k=1}^{n} (n-k)$$

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

Finally, we have

$$E[R] = \sum_{i=1}^{n} \sum_{j=i+1}^{n} E[I_{ij}] = \sum_{i=1}^{n} \sum_{j=i+1}^{n} \frac{1}{2} = \frac{n(n-1)}{4}$$

□

(b) Recall the insertion sort:

```
for i = 1 to n
    j = i
    while j > 0 and A[j-1] > A[j]
        swap A[j] and A[j-1]
        j = j - 1
```

Suppose that our array has d reversals. Prove that the running time of insertion sort is $O(n+d)$.

Proof. The swap operation only occurs when there is an inversion in the array, and each time a swap is performed, the number of inversions decreases by 1. Thus, the total number of swaps is equal to d. Since the outer loop must run n times, the total runtime is $O(n+d)$, as desired. □

(c) What does this imply about the average case running time of insertion sort as a function only of n? That is, if we draw a permutation uniformly at random, what is the expected running time of insertion sort?

Solution. Since $E[R] = \frac{n(n-1)}{4} = O(n^2)$ is the expected number of inversions for a random permutation, the expected running time is $O(n^2+n) = O(n^2)$. □

2 Costly Median

Suppose that you are given n distinct numbers $x_1, x_2, \cdots, x_n \in \mathbb{R}^+$, each of which also has a cost $c_i \in \mathbb{R}^+$ so that $\sum_{i=1}^n c_i = 1$. the costly median is defined to be the number x_k such that

$$\sum_{i:x_i<x_k} c_i < \frac{1}{2} \quad \text{and} \quad \sum_{i:x_i>x_k} c_i \leq \frac{1}{2}$$

Give a deterministic algorithm which finds the costly median and has $O(n)$ worst-case running time (and prove correctness and running time).

Solution. Consider the algorithm:

1. If $n = 0$ or 1, return.
2. Use BPFRT to find the median m of S in $O(n)$ time.
3. Using m, partition the array into L and G, the elements less than, and greater than m.
4. Compute the costs C_L and C_G associated with L and G, respectively.
   (i) If $C_L < \frac{1}{2}$ and $C_G \leq \frac{1}{2}$, then return m.
   (ii) If $C_L < \frac{1}{2}$ and $C_G > \frac{1}{2}$, then the costly median is in G. Add C_L to the cost of m and then recurse on $G \cup \{m\}$.
   (iii) If $C_L \geq \frac{1}{2}$ and $C_G \leq \frac{1}{2}$, then the costly median is in L. Add C_G to the cost of m and then recurse on $L \cup \{m\}$.

Correctness:

Proof. This algorithm always terminates because in step 4, the size of the array we work on always decreases. Now, since the costly median must be in S, it must be in either L or G, and the algorithm operates on whichever array contains the costly median. Thus, the costly median is always in the array during iterations of the algorithm, and the size of the array is eventually trivial, so this must be the correct costly median. ■

Running time:

Proof. Let $T(n)$ be the running time on a set of size n. BPFRT takes at most $10cn$, and splitting the array into L and G costs $n-1$ comparisons. Computing the costs of L and G takes $n-1$ additions. Then in the worst case, we will need to recurse on either L or G, which both have $\frac{n}{2}$ elements. Thus, we have

$$T(n) \leq 10cn + (n-1) + (n-1) + T\left(\frac{n}{2}\right) = (10c+2)n - 2 + T\left(\frac{n}{2}\right)$$

We claim that $T(n) \leq (20c+4)n$. The base case is obviously true. Now,

$$\begin{aligned} T(n) &\leq (10c+2)n - 2 + T\left(\frac{n}{2}\right) \leq (10c+2)n - 2 + (10c+2)n \\ &= (20c+4)n - 2 \leq (20c+4)n \end{aligned}$$

Thus, $T(n) = O(n)$, as desired. ■

□

3 Dumbbell Matching

You belong to a gym which has two sets of dumbbells A and B, each of which has n dumbbells. You know that there is a correspondence between the sets: for every dumbbell in set A there is exactly one dumbbell in set B that has the same weight, and similarly for every dumbbell in set B there is exactly one dumbbell in set A that has the same weight. You want to perform exercises that require two dumbbells of the same weight. So you want to pair up the dumbbells by weight, i.e., for every dumbbell you want to know which dumbbell from the other set has the exact same weight.

Unfortunately the dumbbells are unsorted and unlabeled, and you can't tell their weights by looking at them. The only way to compare two dumbbells is to pick them both up simultaneously (one in each hand) and perform a curl. By comparing the strain on your arms, you can tell whether the two dumbbells are the same weight, and if not, which one is heavier. Even more unfortunately, the owner of the gym has a rule that two dumbbells from the same set cannot be used at the same time. So you can compare a dumbbell from set A to a dumbbell from set B, but cannot compare two dumbbells from the same set.

Design a randomized algorithm which correctly determines the pairing between the sets and uses only $O(n \log n)$ comparisons in expectation. As always, prove correctness and running time.

Solution. Consider the following algorithm:

1. If $n = 0$ or 1, return.
2. Uniformly randomly select a "pivot" dumbbell p from set A.
3. Using p, partition B into 3 sets: the set of dumbbells lighter than p as L, the set of dumbbells heavier than p as H, and the single dumbbell equal in weight to p as q. The rank of p is $|L|+1$, so move it and q to that location, and reorder the dumbbells of B accordingly so dumbbells in L come before q, and dumbbells in H come after.
4. Using q, repeat step 3 to partition and reorder set A, excluding the comparison to p. Now p and q should be in the right spots.
5. Recurse on L and H.

Correctness:

Proof. This algorithm must terminate because L and H are both strictly smaller than the original sets. At each call to the algorithm, the pivot p and its corresponding q must be put in the correct spot. Thus, at conclusion, every dumbbell must be in the correct position. ■

Running time:

Proof. Let a_i and b_i be the ith lightest dumbbells from sets A and B, respectively for $i = \{1, 2, \cdots, n\}$. When we pick a pivot a_p, we perform n comparisons against set B, and then using b_p, we perform $n-1$ comparisons against set A. Then we perform the algorithm on pieces of size $p-1$ and $n-p$. Since each p is equally likely, we have

$$\begin{aligned}
T(n) &= (2n-1) + \sum_{i=1}^{n} P(a_i \text{ is pivot}) E[\#\text{ comparisons} \mid a_i \text{ is pivot}] \\
&= (2n-1) + \sum_{i=1}^{n} \frac{1}{n} \cdot [T(i-1) + T(n-i)] \\
&= (2n-1) + \frac{1}{n}\left(\sum_{i=1}^{n} T(i-1) + \sum_{i=1}^{n} T(n-i)\right) \\
&= (2n-1) + \frac{1}{n} \cdot 2\sum_{i=1}^{n-1} T(i) = (2n-1) + \frac{2}{n}\sum_{i=1}^{n-1} T(i)
\end{aligned}$$

We wish to show that $T(n) \leq cn(\ln n + 1)$ for some $c \geq 1$. The base case $T(1)$ is

$$T(1) = (2 \cdot 1 - 1) + \frac{2}{1}\sum_{i=1}^{0} T(i) = 1 \leq c \cdot 1(\ln 1 + 1) = c$$

which is satisfied. Then we have

$$\begin{aligned}
T(n) &= (2n-1) + \frac{2}{n}\sum_{i=1}^{n-1} T(i) \leq (2n-1) + \frac{2}{n}\sum_{i=1}^{n-1} ci(\ln i + 1) \\
&= (2n-1) + \frac{2c}{n}\left(\sum_{i=1}^{n-1} i \ln i + \sum_{i=1}^{n-1} i\right) \\
&= (2n-1) + c(n-1) + \frac{2c}{n}\sum_{i=1}^{n-1} i \ln i \\
&\leq (2n-1) + c(n-1) + \frac{2c}{n}\int_{1}^{n} x \ln x \, dx \\
&= (2n-1) + c(n-1) + \frac{2c}{n}\left(\frac{1}{2}x^2 \ln x - \frac{1}{4}x^2\right)_{1}^{n} \\
&= (2n-1) + c(n-1) + \frac{2c}{n}\left(\frac{1}{2}n^2 \ln n - \frac{1}{4}n^2 + \frac{1}{4}\right) \\
&= (2n-1) + c(n-1) + cn\ln n - \frac{cn}{2} + \frac{c}{2n} \\
&= cn(\ln n + 1) + \left[2n - 1 - c - \frac{cn}{2} + \frac{c}{2n}\right] \\
&\leq cn(\ln n + 1)
\end{aligned}$$

as long as $c \geq 4$. Thus, $T(n) \leq cn(\ln n + 1) = O(n \log n)$, as desired. ■

□