

CMSC498K Homework 1 Solutions

Richard Matthew McCutchen

Problem 1

I'm going to prove only that $L_{i,j}$ and $M_{i,j}$ are lower and upper bounds on the number of elements greater than the j th in the sample (and not that the bounds are achievable) because this is all that is needed for the rest of the proof. In fact, $M_{i,s}$ is not achievable for $i > 1$: it is $(s + i - 1) \cdot 2^i$ but there are only $s \cdot 2^i$ elements in the entire set!

Lemma: $L_{i,j} = j \cdot 2^i - 1$ and $M_{i,j} = (i + j - 1)2^i$.

Proof: Let $S = (S_1, \dots, S_s)$ be a sample at level i . To avoid clunkiness in stating the proof of the formula for $M_{i,j}$, we begin by observing that the formula is equivalent to saying that S 's set has at least $M'_{i,j} = s \cdot 2^i - M_{i,j} - 1 = (s + 1 - i - j)2^i - 1$ elements less than S_j . The proof proceeds by induction on i .

$i = 0$: A sample at level 0 is just the original set, so the elements greater than the j th element of the sample are exactly elements 1 through $j - 1$ of the sample. Thus $L_{0,j} = M_{0,j} = j - 1$, which agrees with the formulas.

$i > 0$: Suppose the lemma holds for $i - 1$, and let A and B be the samples at level $i - 1$ from which S was generated. S consists of the even-indexed elements of A and B , sorted in increasing order. Without loss of generality, suppose S_j came from A ; specifically, suppose it was A_{2p} . Since A is sorted, even-indexed elements of A appear in S before or after S_j as they appeared before or after A_{2p} in A . Thus, S_1 through S_j consist of A_2, A_4, \dots, A_{2p} as well as some elements of B . Since the elements of A take up p spots, the remaining $j - p$ spots must contain $B_2, \dots, B_{2(j-p)}$. In order for the elements of A to fit, we have $p \leq j$.

Now, what elements of S 's set are necessarily greater than S_j ? By the inductive hypothesis, A 's set has at least $L_{i-1,2p} = 2p \cdot 2^{i-1} - 1$ elements greater than $A_{2p} = S_j$. Furthermore, if $p < j$, then $B_{2(j-p)}$ and the at least $L_{i-1,2(j-p)} = 2(j-p) \cdot 2^{i-1} - 1$ greater elements of B 's set are also greater than S_j , contributing $2(j-p) \cdot 2^{i-1}$ additional elements. (If $p = j$, this quantity is just zero.) Thus, S 's set has a total of at least

$$(2p \cdot 2^{i-1} - 1) + 2(j-p) \cdot 2^{i-1} = 2j \cdot 2^{i-1} - 1 = j \cdot 2^i - 1 = L_{i,j}$$

elements greater than S_j , as desired.

The argument for $M'_{i,j}$ is similar. Based on the composition given above of elements S_1 through S_j , elements S_{j+1} through S_s must consist of $A_{2(p+1)}$ through A_s and $B_{2(j-p+1)}$ through B_s . In order for the elements of A to fit, we have $(s - 2p)/2 \leq s - j$, which is to say, $2(j-p) \leq s$.

What elements of S 's set are necessarily less than S_j ? A 's set contributes at least

$$M'_{i-1,2p} = (s + 1 - (i - 1) - 2p)2^{i-1} - 1$$

such elements. Furthermore, if $2(j-p) < s$, then $B_{2(j-p+1)}$ and the at least

$$M'_{i-1,2(j-p+1)} = (s + 1 - (i - 1) - 2(j - p + 1))2^{i-1} - 1$$

lesser elements of B 's set are also less than S_j , contributing $(s + 1 - (i - 1) - 2(j - p + 1))2^{i-1}$ additional elements. Thus, S 's set has a total of at least

$$((s + 1 - (i - 1) - 2p)2^{i-1} - 1) + ((s + 1 - (i - 1) - 2(j - p + 1))2^{i-1})$$

$$= (2(s+1-(i-1)-p-(j-p+1))2^{i-1}-1 = (s+1-i-j)2^i-1 = M'_{i,j}$$

elements less than S_j , as desired. (If on the other hand $2(j-p) = s$, which is to say $2p = 2j - s$, then the elements of A 's set are enough by themselves:

$$\begin{aligned} (s+1-(i-1)-2p)2^{i-1}-1 &= (s+1-(i-1)-2j+s)2^{i-1}-1 \\ &= (2s+2-i-2j)2^{i-1}-1 \geq (2s+2-2i-2j)2^{i-1}-1 = (s+1-i-j)2^i-1 = M'_{i,j}. \end{aligned}$$

□

Problem 2

We use the trick described in class on February 19 to handle duplicates (assuming, say, that everything in Y comes after everything in X), so from here on we can assume there aren't any. $n = 1$ is trivial; we consider $n \geq 2$. Let $X(i)$ denote the i th element of X .

- Set $i \leftarrow 1$, $j \leftarrow 1$, and $w \leftarrow n$.
- While $w > 2$:
 - Let $k = \lfloor (w-1)/2 \rfloor$. If $X(i+k) < Y(j+k)$, then set $i \leftarrow i+k$; otherwise, set $j \leftarrow j+k$. Either way, set $w \leftarrow w-k$.
- Return the median of $X(i)$, $X(i+1)$, $Y(j)$, and $Y(j+1)$.

Lemma: Let

$$S = \{X(i), \dots, X(i+w-1), Y(j), \dots, Y(j+w-1)\}.$$

Between executions of the loop, the algorithm maintains the invariant that the median of S is the same as that of all $2n$ elements.

Proof: The invariant holds at the beginning because S contains all $2n$ elements. We must show that a single iteration of the loop preserves it. By the inductive hypothesis, it is enough to show that the median of S after the iteration is the same as the median of S before the iteration.

S contains $2w$ elements, and the median is the average of the two middle elements, which have ranks w and $w+1$. WLOG, suppose $X(i+k) < Y(j+k)$; otherwise swap X with Y and i with j in the following argument.

Let $u \in [0, k)$. All the elements $X(i+k)$ through $X(i+w-1)$ are greater than $X(i+u)$ because X is sorted. But we also know $Y(j+k) > X(i+k)$, and elements $Y(j+k+1)$ through $Y(j+w-1)$ are greater than $Y(j+k)$ because Y is sorted. Thus, the $2(w-k)$ elements

$$X(i+k), \dots, X(i+w-1), Y(j+k), \dots, Y(j+w-1)$$

are all greater than $X(i+u)$, so the rank of $X(i+u)$ in S is at most $2w-2(w-k) = 2k \leq w-1$. Therefore, by setting $i \leftarrow i+k$ and $w \leftarrow w-k$, we are effectively dropping the k elements $X(i)$ through $X(i+k-1)$ from S , all of which have rank at most $w-1$ and are therefore less than the two elements determining the median.

Similarly, let $v \in [w-k, w)$. All the elements $Y(j)$ through $Y(j+w-k-1)$ are less than $Y(j+v)$ because Y is sorted. We have $2k \leq w-1$, so $k \leq w-k-1$, so $Y(j+k)$ is among these elements. We also know $X(i+k) < Y(j+k)$, and elements $X(i)$ through $X(i+k-1)$ are less than $X(i+k)$ because X is sorted. Thus, the $(w-k) + (k+1) = w+1$ elements

$$Y(j), \dots, Y(j+w-k-1), X(i), \dots, X(i+k)$$

are all less than $Y(j+v)$, so the rank of $Y(j+v)$ in S is at least $w+2$. Therefore, by setting $w \leftarrow w-k$ and leaving j unchanged, we are effectively dropping the k elements $Y(j+w-k)$ through $Y(j+w-1)$ from S , all of which have rank at most $w+2$ and are therefore less than the two elements determining the median.

We dropped k elements on each side of the two determining the median, so the median remains unchanged, as desired. \square

When the final step is reached, w will be 2, so by the Lemma, the median of $X(i)$, $X(i+1)$, $Y(i)$, and $Y(i+1)$ is the same as that of the original $2n$ elements. Therefore, the algorithm's result is correct. For the running time, note that the quantity $v = w - 2$ decreases by at least half its value on each iteration until it reaches zero (because $k = \lfloor (w-1)/2 \rfloor = \lceil (w-2)/2 \rceil = \lceil v/2 \rceil$), so there are $O(\log v_{orig}) = O(\log n)$ iterations, and each iteration takes constant time.

Problem 3

Here's the incredibly slick algorithm:

- Select the median m of S using the linear-time algorithm based on groups of 5. (If $|S|$ is even, we can just select each of the two middle elements and average them.)
- Create a new array containing the absolute difference between m and each element of S . Any ties in comparing elements of this array are broken by considering earlier elements to be smaller.
- Select the element of rank k in the difference array; call it d .
- Scan the difference array, and for each element less than d as well as d itself, output the corresponding element of S .

Since d has rank k , the algorithm outputs the elements of S corresponding to the k smallest absolute differences. These are just the k elements of S closest to the median, so the algorithm is correct. And for the running time, each of the four steps of the algorithm runs in $O(n)$ time, so the entire algorithm runs in $O(n)$ time.

Problem 4

Whatever number the first die shows, the probability that the second die differs from it is $5/6$. Assuming that happens, the probability that the third die differs from the first two is $4/6$, and so on. Thus, the probability that all the numbers are different is $5!/6^5 = 5/324$.

Problem 5

Let m be the number of edges in the graph. We assume that the vertices are numbered 1 to n and we have the graph in adjacency-list form. Here's the algorithm:

- For each vertex, sort its adjacency list in increasing order.
- For each edge $e = (v_i, v_j)$:
 - Scan the (sorted) adjacency lists of v_i and v_j in tandem, visiting elements in increasing order, as one would do in mergesort. For each vertex v_k that appears in both lists, report (v_i, v_j, v_k) as a triangle.

Clearly every triple (v_i, v_j, v_k) that the algorithm reports as a triangle really is a triangle because (v_i, v_j) is an edge and v_k is adjacent to both of its endpoints. Just as clearly, every triangle in the graph will be reported (once for each of its edges). Thus, the algorithm is correct; we just need to analyze its running time.

Suppose the vertex degrees do not exceed Δ . Then each adjacency list is at most Δ in length, so sorting one adjacency list takes time $O(\Delta \log \Delta)$ and sorting all of them takes time $O(n\Delta \log \Delta)$. Scanning the adjacency lists for one of the m edges takes time proportional to the total length of the two lists, which is $O(\Delta)$. Thus, the entire list-scanning phase takes time $O(m\Delta)$ for a grand total of $O(\Delta(n \log \Delta + m))$.

If Δ is constant, then $m \leq n\Delta/2$ (because the sum of the vertex degrees is at most $n\Delta$ and each edge contributes 2 to this sum), so the running time is linear in n . In the general case, no vertex degree can exceed $n - 1$, so we can take $\Delta = n - 1$ for a running time of $O(n^2 \log n + mn)$.

For extremely sparse graphs ($m = o(n \log n)$), we can improve the running time by leaving the adjacency lists unsorted and processing an edge by inserting the neighbors of each endpoint into a hashtable. This eliminates the sorting phase, and the scanning phase still runs in $O(m\Delta)$ time unless we have really bad luck with the hashtable, so the running time becomes $O(mn)$. (We can avoid the need for a potentially slow reinitialization of the hashtable for each edge by storing a generation counter in each hashtable entry and ignoring entries from outdated generations.)