CSE 4502/5717 Big Data Analytics. Fall 2024

Exam I Solutions

1. Consider the following algorithm:

   repeat
   Pick a random element x from A;
   Perform a binary search in B to check if $x \in B$;
   If $x \in B$, output x and quit;
   forever

   Analysis: Probability that a randomly picked element x of A is in B is $\frac{n^{2/7}}{n}$. This means that the probability of x not being in B is $= 1 - \frac{1}{n^{5/7}}$. As a result, probability that none of the first randomly picked k elements of A is in B is $= \left(1 - \frac{1}{n^{5/7}}\right)^k \leq \exp\left(-\frac{k}{n^{5/7}}\right)$. RHS will be $\leq n^{-\alpha}$ if $k \geq \alpha n^{5/7} \log_e n$. For every element picked from A, we perform a binary search in B that will cost $O(\log n)$ time. Put together, the run time of the algorithm is $\widetilde{O}(n^{5/7} \log^2 n)$.

2. Here is an algorithm:

   Pick a random sample S from A. Let $k = |S|$.
   if A has at least one 0 **then** output "Type 1";
   else if A has at least one 2, **then** output "Type 2";
   else output "Type 1".

   Analysis: Clearly, the run time of the algorithm is $O(k)$.

   If the sample S has a 0 or a 2, the output of the algorithm will certainly be correct. The output could be incorrect only when the sample S has all ones. As a result, consider the possibility of S having all ones only.
   Case 1: The array is of Type 1: In this case, the output will be correct.
   Case 2: The array is of Type 2: Probability that a random element of A is 1 is $\frac{1}{3}$. Probability that all the k elements in S are ones is $\left(\frac{1}{3}\right)^k$. This probability will be $\leq n^{-\alpha}$ if $k \geq \frac{\alpha \log n}{\log 3}$.

   In summary, the run time of the algorithm is $O(\log n)$.

3. A and B are the two input matrices. Let $C = AB$. $C_{ij} = \sum_{k=1}^{n} A_{ik} * B_{kj}$. For every element in the output matrix C, we assign $q = \frac{n}{\log n}$ processors for its computation. Here is a pseudocode for the algorithm:

   for $1 \leq i, j \leq n$ **in parallel do**
   1. Let the processors assigned to C_{ij} be $P_{ij}^1, P_{ij}^2, \ldots, P_{ij}^q$;
   2. These q processors create create an array $Q_{ij}[1:n]$ such that $Q_{ij}[k] = A_{ik} * B_{kj}$, for $1 \leq k \leq n$;

3. These q processors perform a prefix sums computation on $Q_{ij}[1], Q_{ij}[2], \ldots, Q_{ij}[n]$, and hence calculate $Q_{ij}[1] + Q_{ij}[2] + \ldots + Q_{ij}[n] = C_{ij}$ (as one of the results).

Analysis. In line 2, the array $Q_{ij}[1:n]$ can be created in $O(1)$ time using n processors. This can also be done in $O(\log n)$ time using $\frac{n}{\log n}$ processors (via the slow down lemma). Prefix computation in line 3 can be completed in $O(\log n)$ time using $\frac{n}{\log n}$ processors (as was proven in class). As a result, the entire algorithm takes $O(\log n)$ time. The processor bound is $\frac{n^3}{\log n}$ since we assign $\frac{n}{\log n}$ processors for every output element.

4. Let the input sequence be $X = b_1, b_2, \ldots, b_n$. Form the array $Y[1:n]$ such that $Y[i] = \infty$ if $b_i = 0$ and $Y[i] = i$ if $b_i = 1$, for $1 \leq i \leq n$. This array can be formed in $O(\log n)$ time given $\frac{n}{\log n}$ CREW PRAM processors.

   Now perform a prefix minima computation on $Y[1], Y[2], \ldots, Y[n]$. This can also be done in $O(\log n)$ time using $\frac{n}{\log n}$ CREW PRAM processors. Let the output of this prefix computation be $Z[1], Z[2], \ldots, Z[n]$. Output $Z[n]$.

   Clearly, the algorithm takes $O(\log n)$ time using $\frac{n}{\log n}$ CREW PRAM processors.

5. The idea is to employ the standard merging algorithm to merge X and Y and in the process identify elements that are common to X and Y.

   We start by bringing one block each from X and Y into the core memory. We start merging these blocks. While merging, if two copies of any element are found (one coming from X and the other coming from Y), output this element to an output buffer O (residing in the core memory).

   When we run out of elements from X or Y, we bring the next block from that run into the core memory. The size of O is B. When the output buffer O is full, we write this block in the disk and clear O.

   We continue the above merging process until we have processed all the elements from X or Y. When this happens, the algorithm stops and the disk has $X \cap Y$.

   Clearly, we perform at most one pass through X and Y. Thus the I/O complexity of the entire algorithm is $O\left(\frac{n}{B}\right)$.

6. With one pass through X, form another sequence $Y = (q - k_1), (q - k_2), \ldots, (q - k_n)$ and write Y into the disk. Let $Z = X, Y$. Note that Z is a sequence of length $2n$. Sort the sequence Z using the algorithm we discussed in class. Let the sorted sequence be Z'. This will take $O\left(\frac{n}{B}\frac{\log(n/M)}{\log(M/B)}\right)$ I/O operations. Now do one pass through Z' and check if Z' has any repeated elements. If so, output "Yes" else output "No". Note that if there are two elements k' and k" in X whose sum is q, then there will be an element $q - k' = k$" in Y and there will be two copies of k" in Z.