CSE 4502/5717 Big Data Analytics. Fall 2021

Exam I Solutions

1. Consider the following algorithm:

```
repeat
    Pick a random $i \in [1, n]$;
    Perform a binary search in $B$ to check is $A[i]$ is in $B$;
    if $A[i]$ is in $B$, then output $A[i]$ and quit;
forever;
```

Analysis: Call one run of the **repeat** loop as a basic step. Probability of success in one basic step is $\frac{n^{3/4}}{n} = \frac{1}{n^{1/4}}$. Probability of failure in one basic step is $\left(1 - \frac{1}{n^{1/4}}\right)$. Probability of failure in k basic steps is $= \left(1 - \frac{1}{n^{1/4}}\right)^k = \left(1 - \frac{1}{n^{1/4}}\right)^{n^{1/4}\frac{k}{n^{1/4}}} \leq \exp(-k/n^{1/4})$. This probability will be $\leq n^{-\alpha}$ if $k \geq n^{1/4}\log_e n$. This implies that the number of basic steps taken by the above algorithm is $\widetilde{O}(n^{1/4}\log n)$. Each basic step involves a binary search in B that takes $O(\log n)$ time. Put together, the time of the algorithm is $\widetilde{O}(n^{1/4}\log^2 n)$.

To solve this problem deterministically, we can merge A and B in $O(n)$ time and scan through the merged sequence and identify all the common elements in $O(n)$ time.

2. Here is an algorithm:

```
$S = \emptyset$;
for $i = 1$ to $k$ do
    Pick $q$ random elements from $X_i$ and add them to $S$;
Find and output the maximum $M$ of $S$;
```

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

Consider any one specific X_i $(1 \leq i \leq k)$. Probability that the median of X_i is larger than M is $\leq 2^{-q}$. Probability that there exists an i such that the median of X_i is $\geq M$ is $\leq k2^{-q}$. We want this to be $\leq (kn)^{-\alpha}$. This happens when $q \geq \alpha \log n + \log k$.

Therefore, the run time of the algorithm is $O(k(\log n + \log k))$.

3. We will initialize an array $M[1 : 10n]$ in common memory with all zeros.

```
1) Initialize Result to "No";
2) for $i = 1$ to $n$ in parallel do
3)       Processor $i$ writes $A[i]$ in $M[A[i]]$;
4) for $i = 1$ to $n$ in parallel do
```

```
5)      Processor $i$ checks to see if $M[B[i]]$ is zero;
          If not, it tries to write “Yes” in Result;
```

Analysis: The array M can be initialized in one time unit using $10n$ processors. Using the slow-down lemma, this can also be done in $O(1)$ time using n processors. Step 1 takes one unit of time. Steps 3 and 5 take $O(1)$ time each using n processors. Thus the result follows.

4. Here is an algorithm:

```
1) An array $D[1:m]$ is initialized to all zeros.
     The intersection will be written in $I[1], I[2], \ldots$;
2) for $i = 1$ to $m$ in parallel do
3)      Processor $i$ performs a binary search in $A[1:n]$ to see if $B[i] \in A$;
          if $B[i] \in A$, then processor $i$ sets $D[i]$ to 1;
4) All the $m$ processors perform a prefix sums computation on $D[1], D[2], \ldots, D[m]$;
     Let the prefix sums be $E[1], E[2], \ldots, E[m]$;
5) for $i = 1$ to $m$ in parallel do
6)      if $B[i] \in A$, then processor $i$ writes $B[i]$ in $I[E[i]]$;
```

Analysis: Step 1 takes $O(1)$ time. Step 3 takes $O(\log n)$ time. Step 4 takes $O(\log m)$ time. Step 6 takes $O(1)$ time. Thus the total run time is $O(\log n)$.

5. In the main memory we keep C output buffers, one for each possible value that the keys can take. In the disk we grow C runs. Run R_i will be the sequence of all the input keys that have a value i, for $1 \le i \le C$. The algorithm proceeds as follows:

```
1) Bring the next block $X$ from the disk;
2) for each key $x$ in $X$ do
3)      Place $x$ in output buffer $x$;
4) If any of the output buffers is full, write it in the disk;
     Specifically, if output buffer $i$ is full, write this at the end of run $R_i$;
```

After processing all the input keys in the above manner (in one pass through the data), we would have grown C runs that are sitting in the disk. In another pass, we read the runs in order and form one sorted sequence in the disk. The total number of passes is thus two.

6. Keep a data structure Q in memory that supports the following operations: $insert(x)$, $delete(x)$, $findmin()$, $findmax()$, and $search(x)$, where x is an arbitrary element. For example, we can use a 2-3 tree or a red-black tree. Each operation will take $O(\log n)$ time, where n is the number of elements in the data structure.

Read one block at a time from the disk and insert the first $\frac{M}{10}$ elements into Q. After this, do the following:

```
Bring the next block $X$ from the disk;
**for** each element $x$ in $X$ **do**
    $y = findmax(Q)$; **if** $x < y$ **then** $delete(y)$; $insert(x)$;
```

After processing all the input keys in the above manner (in one read pass), output the $\frac{M}{10}$ elements in Q.