CSE 4502/5717 Big Data Analytics; Spring 2026
Exam 1 Helpsheet

1. **Randomized Algorithms.** A Monte Carlo algorithm runs for a prespecified amount of time and its output is correct with high probability. By high probability we mean a probability of $\geq 1 - n^{-\alpha}$, for any constant α (n being the input size). A Las Vegas algorithm always outputs the correct answer and its run time is a random variable. We say the run time of a Las Vegas algorithm is $\widetilde{O}(f(n))$ if the run time is $\leq c\alpha f(n)$ for all $n \geq n_0$ with probability $\geq (1 - n^{-\alpha})$ for some constants c and n_0.

   For the repeated element identification problem we devised a Las Vegas algorithm with a run time of $\widetilde{O}(\log n)$. Given a sequence of n elements, we presented a Monte Carlo algorithm to find an element $\geq$ the median that runs in time $O(\log n)$. We also showed that sorting can be done in $n \log n + \widetilde{O}(n \log \log n)$ comparisons (using the idea of Frazer and McKellar).

2. **Master theorem.** Consider the recurrence relation: $T(n) = aT(n/b) + f(n)$, where $a \geq 1$ and $b > 1$ are constants. **Case1:** If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$. **Case2:** If $n^{\log_b a} = \Theta(f(n))$, then $T(n) = \Theta(f(n) \log n)$. **Case3:** If $f(n) = \Omega(n^{\log_b a + \epsilon})$for some constant $\epsilon > 0$ and $af(n/b) \leq cf(n)$ for some constant $c < 1$, then, $T(n) = \Theta(f(n))$.

3. **Parallel Algorithms**. The model we used was the PRAM (Parallel Random Access Machine). Processors communicate by writing into and reading from memory cells that are accessible to all. Depending on how read and write conflicts are resolved, there are variants of the PRAM. In an Exclusive Read Exclusive Write (EREW) PRAM, no concurrent reads or concurrent writes are permitted. In a Concurrent Read Exclusive Write (CREW) PRAM, concurrent reads are permitted but concurrent writes are prohibited. In a Concurrent Read Concurrent Write (CRCW) PRAM both concurrent reads and concurrent writes are allowed. Concurrent writes can be resolved in many ways. In a Common CRCW PRAM, concurrent writes are allowed only if the conflicting processors have the same message to write (into the same cell at the same time). In an Arbitrary CRCW PRAM, an arbitrary processor gets to write in cases of conflicts. In a Priority CRCW PRAM, write conflicts are resolved on the basis of priorities (assigned to the processors at the beginning).

   We presented a Common CRCW PRAM algorithm for finding the Boolean AND of n given bits in $O(1)$ time. We used n processors. As a corollary we gave an algorithm for finding the minimum (or maximum) of n given arbitrary real numbers in $O(1)$ time using n^2 Common CRCW PRAM processors.

   We also discussed an optimal CREW PRAM algorithm for the prefix computation problem. This algorithm uses $\frac{n}{\log n}$ processors and runs in $O(\log n)$ time on any input of n elements. (For the prefix computation problem the input is a sequence of elements from some domain Σ: $k_1, k_2, \ldots, k_n$ and the output is another sequence: $k_1, k_1 \oplus k_2, \ldots, k_1 \oplus k_2 \oplus k_3 \oplus \cdots \oplus k_n$, where $\oplus$ is any binary associative and unit-time computable operation on Σ.) As an application of prefix computation, we proved that sorting of n elements can be done in $O(\log n)$ time using $\frac{n^2}{\log n}$ CREW PRAM processors.

   The slow-down Lemma: If $\mathcal{A}$ is a parallel algorithm that uses P PRAM processors and runs in T time, then $\mathcal{A}$ can be run on a P'-processor machine to get a run time of T' such that $T' = O\left(\frac{PT}{P'}\right)$, for any $P' \leq P$.

4. **Out-of-core Computing**. In an out-of-core computing model we typically measure only the number of I/O operations (i.e., the I/O complexity) performed by any algorithm. Computing time normally is much less than the I/O time. We let M and B denote the size of the core memory and the block size, respectively. We showed the following results for a single disk model: 1) We can sort N elements with $O\left(\frac{N}{B}\frac{\log(N/M)}{\log(M/B)}\right)$ I/O operations. We first formed runs of length M each and then merged these N/M runs using a M/B-way merge algorithm; 2) There exists a deterministic algorithm for selection whose I/O complexity is $O(N/B)$. BFPRT algorithm was used to achieve this result.

5. We proved that we can find the MST for any given weighted undirected graph $G(V, E)$ in $O\left(\frac{|E|}{B} + |V|\right)$ I/O operations if $M = \Theta(|V|)$.

CSE 4502/5717 Big Data Analytics
Spring 2026 Exam 2 Helpsheet

1. We analyzed the I/O complexity of Prim's algorithm for finding the minimum spanning tree of a weighted graph $G(V, E)$. Assuming that $M = \Theta(|V|)$, we showed that the I/O complexity of Prim's algorithm was $O\left(\frac{|E|}{B} + |V|\right)$.

2. In a Parallel Disks Model (PDM) there are D disks. In one parallel I/O we can bring a block (of size B) of elements from each of the disks. We typically assume that M is a constant multiple of DB. We briefly described the DSM and SRM algorithms for sorting on the PDM. We then introduced the (ℓ, m)-merge sort (LMM) algorithm and showed that it can be used to sort N given elements in no more than $\left\lceil \frac{\log(\frac{N}{M})}{\log(\min\{\sqrt{M}, \frac{M}{B}\})} + 1 \right\rceil^2$ number of passes through the data.

3. Suffix tree is a powerful data structure that can be used to perform a variety of operations on strings and much more. We showed the following results: 1) Given a text T and a pattern P we can search for P in T in $O(m + n)$ time where $m = |T|$ and $n = |P|$; 2) Given a text T and a set $P = \{P_1, P_2, \ldots, P_q\}$ of patterns, we can find all the occurrences of all the patterns in T in $O(m + N + K)$ time where $m = |T|$, N is the total size of all the patterns and K is the total number of occurrences of all the patterns in T; 3) Given a database DB of texts $\{T_1, T_2, \ldots, T_k\}$ and a set of patterns $P = \{P_1, P_2, \ldots, P_q\}$, we can find occurrences of all the patterns in DB in $O(M+N+K)$ time where M is the total size of all the texts in DB, N is the total size of all the patterns, and K is the total number of occurrences of all the patterns in DB; 4) Given two strings S_1 and S_2, we can find the longest common substring between them in $O(|S_1| + |S_2|)$ time; 5) Given two strings S_1 and S_2 and an integer l, we can find all the substrings of S_2 of length $\geq l$ that occur in S_1 in $O(|S_1| + |S_2|)$ time; 6) Given a string S_1, a collection of strings $C_1, C_2, \ldots, C_q$ and an integer l, we can find all the occurrences of C_i of length $\geq l$ in S_1 (for $1 \leq i \leq q$) in $O(|S_1| + \sum_{i=1}^{q} |C_i|)$ time; 7) Given n strings $S_1, S_2, \ldots, S_n$, we can compute $\ell[2], \ell[3], \ldots, \ell[n]$ in $O(Mn)$ time, where $\ell[i]$ is the length of the longest substring that occurs in $\geq i$ input strings $(2 \leq i \leq n)$ and $M = \sum_{i=1}^{n} |S_i|$; and 8) Given n strings of total length M, we can solve the all pairs suffix-prefix problem in $O(M + n^2)$ time.

4. We showed that we can sort n integers in the range $[1, n^c]$ in $O(n)$ time, c being any constant.

5. We can use the suffix array and the longest common prefix (LCP) array to search for a pattern P in a text T in $O(n + \log m)$ character comparisons, where $m = |T|$ and $n = |P|$. We also pointed out that we can compute the LCP array (for pairs of interest in string matching) in $O(m)$ time. We also presented the skew algorithm for constructing a suffix array that takes $O(m)$ time on any input string of length m.

CSE 4502/5717 Big Data Analytics
Spring 2026 Exam 3 Helpsheet

1. **Association Rules Mining.** An **itemset** is a set of items. A k**-itemset** is an itemset of size k. A **transaction** is an itemset. A **rule** is represented as $X \to Y$ where $X \neq \emptyset, Y \neq \emptyset, X \cap Y = \emptyset$.

   We are given a database DB of transactions and the number of transactions in the database is n. Let I be the set of distinct items in the database and let $d = |I|$.

   For an itemset X, we define $\sigma(X)$ as the number of transactions in which X occurs, i.e. $\sigma(X) = |\{T \in DB | X \subseteq T\}|$ The **support** of any rule $X \to Y$ is $\frac{\sigma(X \cup Y)}{n}$. The **confidence** of any rule $X \to Y$ is $\frac{\sigma(X \cup Y)}{\sigma(X)}$.

   Association Rules Mining is defined as follows.

   Input: A DB of transactions and two numbers: minSupport and minConfidence.
   Output: All rules $X \to Y$ whose support is $\geq$ minSupport and whose confidence is $\geq$ minConfidence.

   An itemset is **frequent** if $\sigma(X) \geq n \cdot \text{minSupport}$

   We discussed the Apriori algorithm for finding all the frequent itemsets. This algorithm is based on the a priori principle: If X is not frequent then no superset of X is frequent. Also, If X is frequent then every subset of X is also frequent.

   The pseudocode for the Apriori algorithm is given next.

Algorithm 1: Apriori algorithm

```
k := 1;
Compute F_1 = {i ∈ I | σ(i) ≥ n · minSupport};
while F_k ≠ ∅ do
    k := k + 1;
    Generate candidates C_k from F_{k−1};
    for T ∈ DB do
        for C ∈ C_k do
            if C ⊆ T then
                σ(C) := σ(C) + 1;
    F_k := ∅;
    for C ∈ C_k do
        if σ(C) ≥ n · minSupport then
            F_k := F_k ∪ {C};
```

   We can use a hash tree to compute the support for each candidate itemset.

   We also presented a randomized Monte Carlo algorithm for identifying frequent itemsets. The idea was to pick a random sample, identify frequent itemsets in the sample (with a smaller support) and output these. We proved that the output of this algorithm will be correct with a high probability using the Chernoff bounds:

   If X is $B(n, p)$, then the following are true:

$$Prob.[X \geq (1+\epsilon)np] \leq \exp(-\epsilon^2 np/3)$$

$$Prob.[X \leq (1-\epsilon)np] \leq \exp(-\epsilon^2 np/2),$$

for any $0 < \epsilon < 1$.

2. **Hierarchical Clustering.** We showed that we can perform hierarchical clustering on n given points in $O(n^2)$ time. The idea of hierarchical clustering is to start with n clusters where each input point is a cluster on its own. From thereon, we merge the two closest clusters at a time until a termination condition is reached.

3. **Quantum Computing.** We introduced quantum circuits. We discussed the ideas of entanglement and teleportation. We also stated the no cloning theorem. In addition, we also gave the details of Grover's algorithm. Grover's algorithm solves the following problem: Given a sequence $X = x_1, x_2, \ldots, x_N$ and a function $f : X \to \{0, 1\}$, find an i such that $f(x_i) = 1$. Grover's algorithm solves this problem in $O(\sqrt{N})$ time using a quantum circuit. The output of this circuit will be correct with a constant probability.

4. **Polynomial Arithmetics.** A degree-n polynomial can be evaluated at a given point in $O(n)$ time. Lagrangian interpolation algorithm runs in $O(n^3)$ time whereas Newton's interpolation algorithm takes $O(n^2)$ time.

   Two degree-n polynomials can be multiplied in $O(n \log n)$ time. A degree-n polynomial can be evaluated at n given arbitrary points in $O(n \log^2 n)$ time. Also, interpolation of a polynomial presented in value form at n arbitrary points can be done in $O(n \log^3 n)$ time.