EE608: Final Exam

1. Consider the following two questions about the computational complexity of an algorithm. **(8 points total)**

   (a) If the best-case running time of an algorithm FOO-BAR is $\Theta(n^3)$, what can be said about the average-case and worst-case running time of FOO-BAR? Explain. **(4 points)**

   Since the average-case and worst-case running time of FOO-BAR will be larger than equal to the best-case running time, we can say that the average-case and worst-case running time is $\Omega(n^3)$. Nothing else can be said.

   (b) If the worst-case running time of BAR-FOO is $\Omega(2^n)$, what can be said about the best-case and average-case running time of BAR-FOO? Explain. **(4 points)**

   The worst-case running time of BAR-FOO can be anything that is in $\Omega(2^n)$. Since the best-case and average-case running time can be the same as or faster than the worst-case running time, the best-case and average-case running time of BAR-FOO could be anything.

2. Solve the following recurrences using Master Theorem or explain why Master Theorem can't be used. Provide all necessary details. **(12 points total)**

   (a) $T(n) = 3T(\frac{n}{3}) + n^2$ **(4 points)**

   This recurrence can be solved by using case 3 of the Master Theorem since $a = 3, b = 3$, and $f(n) = n^2$, it follows that $n^2 = \Omega(n^{log_3 3+\epsilon})$ for some $\epsilon > 0$ and $3((n/3)^2) \leq c(n^2)$ for some $c < 1$, say $c = \frac{1}{2}$. Thus, $T(n) = \Theta(n^2)$.

   (b) $T(n) = T(\frac{n}{2}) + \sin n$ **(4 points)**

   Since $f(n) = \sin(n)$, it ranges between $[-1, 1]$. Hence, $f(n)$ is not asymptotically positive and so this recurrence cannot be solved by using Master Theorem.

   (c) $T(n) = 4T(\frac{n}{3}) + 1$ **(4 points)**

   This recurrence can be solved by using case 1 of the Master Theorem since $1 = O(n^{log_3 4-\epsilon})$ for some $\epsilon > 0$. Thus, $T(n) = \Theta(n^{log_3 4})$.

3. Provide a summation expressing for how many times statement 4 is executed in the following code segment. It may help you to consider how often the inner loop iterates for each i value given a small value for n. You do not need to solve the summation in closed form, but you may want to verify that your summation is correct by considering small cases. Give analysis details if you expect partial credit. **(12 points)**

```
1. for $i \leftarrow 1$ to $n$
2.     do $j \leftarrow i$
3.         while $j < n$
4.             do $j \leftarrow j + 4$
```

Let $n = 10$. Then, when $i = 1$, the line 4 executed 3 times when $j = 1, 5, 9$ and the **while** loop terminates when $j = 13$. When $i = 2$, the line 4 executed 2 times when $j = 2, 6$ and the **while** loop terminates when $j = 10$. When $i = 3$, the line 4 executed 3 times when $j = 3, 7$ and the **while** loop terminates when $j = 11$. When $i = 4$, the line 4 executed 2 times when $j = 4, 8$ and the **while** loop terminates when $j = 12$. When $i = 5$, the line 4 executed 2 times when $j = 5, 9$ and the **while** loop terminates when $j = 13$. When $i = 6$, the line 4 executed 1 time when $j = 6$ and the **while** loop terminates when $j = 10$. When $i = 7$, the line 4 executed 1 time when $j = 7$ and the **while** loop terminates when $j = 11$. When $i = 8$, the line 4 executed 1 time when $j = 8$ and the **while** loop terminates when $j = 12$. When $i = 9$, the line 4 executed 1 time when $j = 9$ and the **while** loop terminates when $j = 13$. When $i = 10$, the line 4 is not executed because the **while** loop terminates when $j = 10$.

From this simple example, we can see that when $i = k$, the inner loop executes $\lceil \frac{n-k}{4} \rceil$, so Line 4 is executed $\sum_{i=1}^{n} \lceil \frac{n-i}{4} \rceil$.

4. Consider the following questions about sorting. **(8 points total)**

   (a) If the first line of RADIX-SORT is changed to **for** $i \leftarrow d$ **down to** 1, will the algorithm still be correct? Explain. **(5 points)**

   The algorithm will not be correct. Consider a simple example, $A = \langle 13, 31 \rangle$. The above approach will yield $\langle 31, 13 \rangle$ as the sorted output, which is not correct.

   (b) Give the main reason why a person would prefer to use RANDOMIZED-QUICKSORT instead of QUICKSORT. What is the worst-case running time of RANDOMIZED-QUICKSORT? **(3 points)**

   We use RANDOMIZED-QUICKSORT to prevent specific inputs from eliciting the worst-case running time for QUICKSORT. The worst-case running time for the randomized version is $\Theta(n^2)$, just as for QUICKSORT.

5. Design an algorithm FIND-PAIR(S, Z) that has an $O(n)$ average-case running time to determine whether there exists a pair of distinct numbers $x, y \in S$, where S is a dynamic set of n integers and Z is a number, such that $x + y = Z$. The algorithm should return the first x, y pair found such that $x + y = Z$, or NIL if no such x, y pair exists in S. You may assume all of the numbers in S are distinct (i.e., no duplicates). Show that you meet the time bound. Half credit will be given for less efficient solutions. **(14 points)**

   (a) Store all n elements of S in a chained hash table of size n, T.

   (b) For each $x \in S$, call CHAINED-HASH-SEARCH$(T, Z - x)$. Since there are no duplicates, when $Z - x = x$, we should not look for a value in the table; simply continue with another value in the set. If CHAINED-HASH-SEARCH$(T, Z - x)$ returns with a value other than NIL, return x, $Z - x$. If a call for a particular x returns NIL, we continue with the next value of x. Search continues until a pair is located or until all of the elements of S have been tried, in which case NIL is returned.

The time to store the elements of S in T is $\Theta(n)$. If there is no $x, Z - x$ pair, then n searches are required, each taking $O(1)$ time on average. Hence, this step will take at most $O(n)$ time on average. Hence, the expected time is at most $O(n)$.

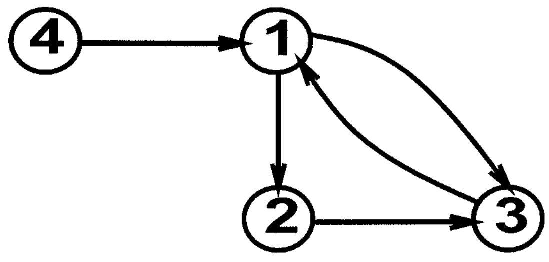
6. Consider a variant of the MATRIX-CHAIN-ORDER multiplication problem in which the goal is to parenthesize the sequence of matrices in order to **maximize** the number of scalar multiplications. Does this problem exhibit the optimal substructure property? If so prove it, if not give a specific counter-example. **(13 points)**

   When multiplying together two matrices, the number of scalar multiplications is based on the number of rows and columns of the first and the number of columns of the second (where the row dimension of the second must be equal to the column dimension of the first). If we wish to optimize so that we maximize the number of scalar multiplications, then the problem still exhibits the optimal substructure property. To see that this is so, assume you are given an optimal parenthesization of $A_{i,j}$, i.e., $A_i \times A_{i+1} \times \ldots \times A_j$ that is comprised of two subproblems $A_{i,k}$ and $A_{k+1,j}$. For $A_{i,j}$ to be optimal, it must be the case that $A_{i,k}$ and $A_{k+1,j}$ are optimal solutions to subproblems. Suppose that there is another solution to $A'_{i,k}$ with more scalar multiplications than $A_{i,k}$, then this would contradict the optimality of $A_{i,j}$ since the number of scalar multiplications resulting from multiplying the two resulting matrices is fixed, so the alternative solution $A'_{i,k}$ would yield a solution with more scalar multiplications than $A_{i,j}$.

7. Prove or disprove that any Minimal Spanning Tree (MST) for a weighted undirected connected graph $G = (V, E)$ containing at least three vertices must contain the second smallest edge. You may assume that all edge weights are distinct. **(13 points)**

   We prove the second smallest edge e must be in any MST using proof-by-contradiction. Assume that there is an MST T which does not contain e. Adding e to T creates a cycle p. The maximum weight edge in p, call it e' must be such that $w(e') > w(e)$ because the cycle must contain at least three edges, only one of which can be smaller than e. We can break the cycle by removing e' to create T' such that $w(T') = w(T) - w(e') + w(e) < w(t)$, contradicting our assumption that T was an MST. Hence, any MST must contain the second smallest edge.

8. List all of the $T^{(i)}$ matrices (for $i = 0$ up to $i = 4$) that are created by the TRANSITIVE-CLOSURE algorithm applied to the graph below. **(5 points)**



$$T^{(0)} = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix}$$

$$T^{(1)} = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

$$T^{(2)} = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

$$T^{(3)} = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

$$T^{(4)} = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

9. Consider the following variation of FLOYD-WARSHALL, called FLOYD-WARSHALL-ALT. **(15 points)**

```
FLOYD-WARSHALL-ALT($W$)
1. $n \leftarrow rows[W]$
2. $D \leftarrow W$
3. for $k \leftarrow 1$ to $n$
4.     do for $i \leftarrow 1$ to $n$
5.         do for $j \leftarrow 1$ to $n$
6.             $d_{ij} \leftarrow min(d_{ij}, d_{ik} + d_{kj})$
7. return $D$
```

(a) What are the space requirements of FLOYD-WARSHALL-ALT? How does it compare to FLOYD-WARSHALL? **(5 points)**

This algorithm requires $\Theta(n^2)$ space compared to $\Theta(n^3)$ used by the original algorithm.

(b) Is FLOYD-WARSHALL-ALT correct? In other words, does FLOYD-WARSHALL-ALT return the same result as FLOYD-WARSHALL? Justify your answer thoroughly. **(10 points)**

FLOYD-WARSHALL-ALT calculates the value for d_{ij} for iteration k by looking up the values of d_{ij}, d_{ik}, and d_{kj} from the D matrix as computed during iteration $k-1$. It must be the case that d_{ij} is unchanged at any other time during the iteration because the algorithm only sets the value after looking up the previous value, and d_{ik} and d_{kj} cannot be changed by introducing k as an intermediate vertex (unless there is a negative weight cycle, in which case both D and $D^{(n)}$ are invalid); hence, these values are available from the prior iteration. Neither algorithm detects negative weight cycles, and so both would require the same modification to handle that case. In all other cases, FLOYD-WARSHALL-ALT is correct.

10. Show that the **Set Cover** (*SC*), which is described below, is NP-complete given that **Vertex Cover** (*VC*) (also described below) is a known NP-complete problem. **(20 points)**.

    Set Cover (*SC*)

    INSTANCE: A family of sets $S_1, S_2, \ldots, S_n$ and an arbitrary positive integer $k \leq n$.

    QUESTION: Is there a subfamily of k sets from the original family of sets, $S_{i_1}, S_{i_2}, \ldots, S_{i_k}$ such that:

    $$\bigcup_{j=1}^{k} S_{i_j} = \bigcup_{j=1}^{n} S_j$$

    Vertex Cover (*VC*)

    INSTANCE: An undirected graph $G = (V, E)$ and an arbitrary positive integer $K \leq |V|$.

    QUESTION: Is there a **vertex cover** of size K or less for G, i.e., a subset $V' \subseteq V$ such that $|V'| \leq K$ and for each edge $(u, v) \in E$, at least one of u and v is in V'?

Given an instance of *SC*, the number of sets in the solution can be counted in linear time to ensure that it is less than or equal to k. It is also easy to verify that each set in the subfamily appears in the original family of sets in polynomial time. Finally, the solution family of sets and the original family of sets can be unioned in polynomial time (e.g., $O(n \lg n)$ worst case time to sort each list and merge removing duplicates) and the resulting sets checked for set equality (in $O(n)$ worst case time). Hence, *SC* is in NP.

We will show that $VC \propto SC$. For any *VC* instance with an undirected graph $G = (V, E)$ and a value K, construct an *SC* instance as follows: for $1 \leq i \leq |V|$, let S_i be the set of all edges incident upon v_i, and let $k = K$. Clearly, this requires only $O(V + E)$ time when G is represented using an adjacency list. Also, if there is a *VC* of size $k = K$ $v_{i_1}, v_{i_2}, \ldots, v_{i_k}$, then clearly the union of the sets $S_{i_1}, S_{i_2}, \ldots, S_{i_k}$ gives precisely the edges of the union of the sets $S_1, S_2, \ldots, S_V$. Furthermore, if there is an *SC* $S_{i_1}, S_{i_2}, \ldots, S_{i_k}$ with $k = K$, then a *VC* exists for G with size $k = K$, namely $v_{i_1}, v_{i_2}, \ldots, v_{i_k}$.