Final Exam – Fall 2020
CS 412: Algorithms: Design and Analysis
Name: Muhammad Ammar
Student ID: ma05168
Section: L1
Filename: ma05168-L1-final-solution.pdf
Start date/time: 9:00A.M.
End date/time: 8:50PM

File name convention: $\langle xxddddd \rangle$-$\langle$section$\rangle$-final-solution.pdf

where $x \in \{a, b, \ldots, z\}^*$, $d \in \{0, 1, \ldots, 9\}^*$, and section $\in$ {L1, L2, L3}. For example, xy01234-L1-final-solution.pdf.

Instructions:

1. Please make sure you have read and understood all instructions from the question paper.
2. Please don't change the structure of this file.
3. Please write your solution in the provided "solution boxes".
4. Please make sure your solution file is named as mentioned above.
5. Please make sure you signed tne following "Honor Pledge".

Honor Pledge

I affirm that I have not used any unfair means to solve this exam. I also affirm that I have not received and/or provided any unauthorized help on this exam, and that all work is by my own.

Signature: Muhammad Ammar

Question 1: [28 points]

For each of the following indicate whether the statement is True or False [**01 point each**]. Justify your answer [**03 points each**].

(a) An unbiased coin is flipped repeatedly until both heads and tails are obtained. The expected number of times the coin is flipped is 3.

Solution: True. If we toss an unbiased coin then the expected value to get a head or tail on first toss is 1, because there would be either head or tail. After first toss, the case becomes a geometric random variable with a probability of $\frac{1}{2}$. Using formula of expected value of geometric $R.V. = \frac{1}{p} = \frac{1}{\frac{1}{2}} = 2$. Hence, adding expected values of both cases $E = 1 + 2 = 3$. Hence, Proved!

(b) If an undirected graph G with n vertices has k connected components then there are at most $n - k$ edges in G.

Solution: False. This is because in the case of n nodes of an undirected complete graph. Since every node is connected to every other node except it self, therefore the number of edges would be $\frac{n(n-1)}{2}$. In this case value of k would be one since there would only one SCC. However, the at most edges in this case is greater than n - 1 i.e. $\frac{n(n-1)}{2}$.

(c) Let T be a minimum spanning tree of a graph G. Then for any pair of vertices s and t in G, the shortest path from s to t in G is the shortest path from s to t in T.

Solution: False. Consider an undirected graph of four vertices such that they form a square. Now in MST, we have to lose an edge between any two vertices in order to avoid forming a cycle. Thus, the shortest path between the nodes whose edge has been removed in MST would be lost. Hence Proved!

(d) Let $G_1 = (V, E)$ and $G_2 = (V, E)$ be two *directed graphs* with the same structure (i.e., same vertices and same edges). Let the costs of edges in G_1 be distinct and nonnegative and costs of edges in G_2 be the squares of costs of their corresponding edges in G_1. Then the *shortest paths* in G_1 and G_2 from some vertex s to any other vertex t by DIJKSTRA's algorithm are the same.

Solution: False. Consider a case where vertex a is connected with b with a weight of 10 and a is also connected to c with a weight of 6 and c is connected to d with a weight of 5. In this case, going $a- > b$ directly costs 10, where as $a- > c- > b$ costs $5 + 6 = 11$. Hence, shortest path is 10. However, after squaring the distances, $a- > b$ would cost 100 and a -¿ c -¿ b costs $25 + 36 = 61$. Thus the shortest path now is $a- > c- > b$. Hence Proved!.

(e) Every directed graph is a *directed acyclic graph* (DAG) of its *strongly connected components* (SCC).

Solution: True. The reason is because there is no possible way of having a directed graph in which there is a cycle other than a SCC. Since, there would not be any cycle we can say that every directed graph is a directed acyclic graph of its strongly connected components.

(f) There exists an $O(n^2)$ algorithm to generate all the possible bit strings of length n.

Solution: False. The number of possible bit strings that could be generated of length n would be 2^n. Thus, to design on algorithm of $O(log(n))$ is not possible. The complexity of this algorithm would be 2^n

(g) The *exact* value returned from the following algorithm MYFUNCTION expressed in terms of n is $n(n+1)/2$.

Algorithm MYFUNCTION

Input: An integer n

Output: The value k

```
1. $k = 0$
2. for $i = 0$ to $n - 1$ do
3.     for $j = i$ downto $0$ do
4.         $k = k + 1$
5. return $k$
```

Solution: True. The algorithm provides the sum of natural numbers, which is equal to $\frac{n(n-1)}{2}$

Question 2: [15 points]

We are given an array $A[1..n]$, which stores a sequence of 0's and then followed by a sequence of 1's.

(a) $\boxed{\text{05 points}}$ Design an $O(\log n)$-time algorithm to find the location of the last 0, i.e., find k such that $A[k] = 0$ and $A[k+1] = 1$.

Solution: The algorithm is given below:
A recursive function which takes an array, and two variables as arguments l and h, initially zero and length of array respectively. It is a binary search on 0 value where the next value is 1. Thus in worst case the complexity would be $O(logn)$.

```
1. def findZero(lst, l, h):
2.     if $high \geq low$ :
3.         $mid = \frac{h+l}{2}$
4.         if $arr[mid] == 0$ and $arr[mid+1] == 1$ :
5.             return mid
6.         else if $arr[mid] == 1$ and $arr[mid+1] == 1$ :
7.             return findZero(lst, l, mid+1)
8.         else:
9.             return findZero(lst, mid+1, high)
```

(b) $\boxed{\text{10 points}}$ Suppose that k is much smaller than n. Is it possible to improve the running time of our algorithm to $O(\log k)$ instead of $O(\log n)$? Justify your answer (i.e., provide an algorithm if your answer is "yes" or a proof if your answer is "no").

Solution: Yes.

```
• $low = 0, high = 1$
•     while $lst[high]1$:
•         $low = high$
•         $high = 2 * high$
• return findZero(lst, low, high)
```

The complexity of above algorithm would be $O(log(k))$. Since, we are jumping with a power of 2, until we reach or pass k which takes time $log(k)$. When found, it will do binary search over that portion of the array which would again on an upper bound would take $log(k)$ times. Hence, the final Big-O Complexity in the worst case would be $O(log(k))$.

Question 3: [10 points]

Let $A[1..n]$ be an array of n distinct numbers. If $i < j$ and $A[i] > A[j]$, then the pair (i, j) is called an *inversion* of A. Suppose that the elements of A form a *uniform random permutation* i.e., $\Pr\{A[i] > A[j]\} = 1/2$ for all $i < j$. Use indicator random variables to compute the expected number of inversions.

Solution: The number of possible inversions is $\binom{n}{2}$ for our case. Therefore, the number of indicator random variables to be defined is $\binom{n}{2}$ where a single indicator random variable X for any i, j s.t $i < j$ is:

$$X_{ij}$$

where,

$$X_{ij} = \begin{cases} 1, & \text{for } A[i] > A[j] \\ 0, & \text{for } A[i] < A[j] \end{cases}$$

Since, the uniform random permutation for any given pair is $\frac{1}{2}$ therefore probability of random variable X is:

$$\mathbb{P}(X_{ij} = 1) = \tfrac{1}{2}$$
$$.$$
$$\mathbb{P}(X_{ij} = 0) = \tfrac{1}{2}$$

Hence, using the formula for expected variable of random variable:

$$\mathbb{E}[X_{ij}] = (1 * \mathbb{P}(X_{ij} = 1)) + (0 * \mathbb{P}(X_{ij} = 0)) = \tfrac{1}{2}$$

For expected value of all the inversions Z, we can say that:

$$\mathbb{E}[Z] = Z$$
$$.$$
$$\mathbb{E}[Z] = \textstyle\sum_{i<j} \mathbb{E}[X_{ij}]$$

where summation would contain $\binom{n}{2} = \frac{n(n+1)}{2}$ terms.

$$\mathbb{E}[Z] = \tfrac{n(n+1)}{2}\tfrac{1}{2}$$
$$.$$
$$\mathbb{E}[Z] = \tfrac{n(n+1)}{4}$$

Question 4: [10 points]

Let $A[1..n]$ be an array of *nonnegative integers* and assume there are exactly k 0's in A, note that $0 < k < n$. Consider the following algorithm. Compute the expected number of times the Step **(3)** is executed.

Input: An array $A[1..n]$ of nonnegative integers and an integer k s.t. $0 < k < n$.
Output: An array $B[1..k]$ of indices of all 0's in A.

```
1. $B = [\,]$
2. while $|B| \neq k$
3.     Generate a random number $j$ in $\{1, \dots, n\}$
4.     if $A[j] = 0$
5.         if $j \notin B$ then append $j$ to $B$
6. return $B$
```

For example if $A = [0, 5, 2, 0, 8, 9, 0, 3, 1, 7, 0]$ then one possible solution could as $B = [4, 11, 7, 1]$ (the elements in B might not be ordered).

Solution: In order to determine the expected value, we need to determine the nature of random variable for this case. Since, there can be "f" number of failures before actually getting an index of zero, i.e. first success therefore this is a case of geometric random variable. Let X_i be our geometric random variable, which represents the i_{th} success where ik. Using the formula of Expected Value of Geometric R.V.,
As we know that,

$$\mathbb{E}[G.R.V] = \tfrac{1}{p}$$

Hence, Expected values of first, second up till $(k-i)^{th}$ Random Variable.

$$\mathbb{E}[X_1] = \tfrac{1}{\frac{k}{n}} = \tfrac{n}{k}$$
$$\mathbb{E}[X_2] = \tfrac{n}{k-1}$$
$$\mathbb{E}[X_3] = \tfrac{n}{k-2}$$
$$\mathbb{E}[X_k] = \tfrac{n}{k-(k-1)}$$

Now, since every expected value for any number of success is independent of other, we can just simply sum all the expected values of X_1 up to X_k to find the expected number execution of step 3:

$$\mathbb{E}[X] = X_1 + X_2 + X_3 ... X_K$$
$$\mathbb{E}[X] = \tfrac{n}{k} + \tfrac{n}{k-1} + \tfrac{n}{k-2} \cdots \tfrac{n}{1}$$
$$\mathbb{E}[X] = \sum_{i=0}^{i=k-1} \tfrac{n}{k-1}$$

Question 5: [12 points]

Given a rooted tree $T = (V, E)$.

(a) $\boxed{\text{5 points}}$ Design an algorithm that checks whether two vertices $u, v \in V$ are at the same level in T or not.

Solution: nodeLevel is a dictionary of size $|V|$ vertices which stores the level of each node. Visited is an array containing information of a node whether it has been visited or not.

```
1. Queue = q
2. add root to q
3. nodeLevel[root] = 0
4. visited[root] = true
5. while Q is not empty:
6.     n = q.dequeue()
7.     loop over all m neighbors of n:
8.         if m not in visited:
9.             level[y] = level[x] + 1
10.            visited[y] = true
11.            q.queue(y)
12. if levels of node $u$ and $v$ is same:
13.     return true
14. else:
15.     return false
```

(b) $\boxed{\text{5 points}}$ What is the time complexity of your algorithm?

Solution: The time complexity would be similar to what is of breadth first algorithm i.e. $O(|V| + |E|)$. Since, BFS is what we're basically doing in this algorithm.

(c) $\boxed{\text{2 points}}$ What is the space complexity of your algorithm?

Solution: Since, the two major things the algorithm is storing is a dictionary which is of size of vertices or less, and dictionary of levels of size of vertices too. Thus, the time complexity would be $O(|V| + |V|) = O(2|V|) = O(|V|)$

Question 6: [15 points]

For a given directed graph $G = (V, E)$ and weight function $w : E \to \mathbb{R} \cup \{\infty\}$ defined for all edges in G. The *diameter* of the graph G is the length of the *longest shortest path* between any two vertices (u, v).

Use *bottom-up dynamic programming* to solve following questions.

(a) $\boxed{\text{5 points}}$ Express the problem recursively showing that an optimal substructure exists.

Solution: Consider a graph of n nodes with vertices array of length n. Let us consider a distance function which returns the shortest distance between two nodes. It takes two nodes as arguments u and v and a number a. The a is the number of intermediary nodes between u and v. Thus, it will only look for a vertices.
It will recursively call for every two nodes, in such a way like:

$$distance(u, v, a-1) =$$
$$min(distance(u, v, a), distance(u, x_a, a-1) + distance(x_a, v, a-1)) \text{ until a = 1.}$$

Diameter = maximum distance of all the distances obtained from distance function.

(b) $\boxed{\text{5 points}}$ Design an $O(|V|^3)$ algorithm that returns the diameter of the graph G.

Solution: Let n be the number of nodes, $V = [V_0, V_1, V_2, ..., V_n]$
E be the adjacency matrix of 2D array with the size of n by n.
R be the 2D array with every element equal to

```
• for i in range(n):
•     for j in range(n):
•         for k in range(n):
•             R[i][j] = min(D[i][j],D[i][k] + D[k][j])
• return the R[i][j] with the minimum value.
```

(c) $\boxed{\text{5 points}}$ Derive the runtime complexity of the algorithm using asymptotic notation.

Solution: Since the algorithm computes the adjacency matrix in the $O(|V|^2)$ and computes the shortest distances in $O(|V|^3)$ and finding out the diameter in $O(|V|^2)$, therefore the complexity of the algorithm would become.

$$O(|V|^2) + O(|V|^3) + O(|V|^2)$$

which is approximately equals to:

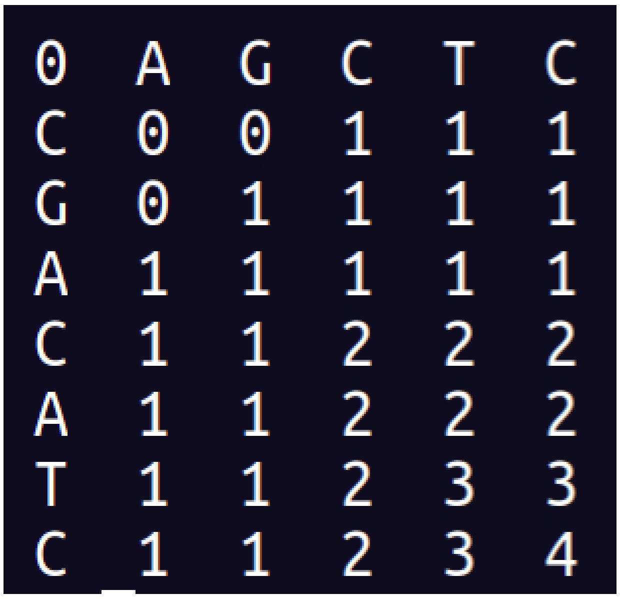
$$O(|V|^2)$$

Question 7: [10 points]

Find a *longest common subsequence* between the two string sequences:

`CGACATC` and `AGCTC`.

Solution: The longest common subsequence between the two string sequences is, ACTC and GCTC.
Memoization technique was used where we make a n by m size where n is the size of first string and m is the size of second. Each index on table represents the longest common subsequence between the two strings from start up till that index. For instance, at index $(3, 5)$ the value is 3. This means that from index 0 to 3 on first string and from 0 to 5 on second string, the longest common subsequence possible is of length three. In order to find for both strings completely we look at the table's bottom right most index i.e. (4, 6).



0	A	G	C	T	C
C	0	0	1	1	1
G	0	1	1	1	1
A	1	1	1	1	1
C	1	1	2	2	2
A	1	1	2	2	2
T	1	1	2	3	3
C	1	1	2	3	4