Question 1

(a) True

(b) False

(c) True

(d) False

(e) True

Question 2

(a)

Level	Nodes per Level	Work per Node	Total Work at Level
0	1	6n^2	6n^2
1	3	6(n/3)^2	18(n/3)^2
2	9	6(n/9)^2	54(n/9)^2
3	27	6(n/27)^2	162(n/27)^2
...	...	...	...
$\log_3(n-2)$	n	2	2n

Input size at level i: n/(3^i)

Work per node at level i: 6(n/(3^i))^2

Number of nodes at level i: 3^i

Total work at level i: 6n^2/(3^i)

Last level of i: $\log_3(n-2)$

Total work at last level: 2n

T(n) $= \sum_{i=0}^{\log_3(n-2)-1}$ 6n^2/(3^i) $+ 2n$

T(n) $= 6n^2 \sum_{i=0}^{\log_3(n-2)-1} 1/(3^i) + 2n$

T(n) $\in \theta(n^2)$

(b) Master Theorem:

$\log_3 3 = 1$

$n^2 \rightarrow 2$. $1 < 2$, so T(n) $\in \theta(n^2)$

Question 3

Statement: "If Dijkstra's is run on a graph with no negative edge weights: at the start of the ith iteration of the while loop, if u is the closest unprocessed vertex, u.dist stores the shortest path distance from source to u for all i¿=0". We will make the assumption that the graph given is not empty so the source vertex exists and there are no negative edge weights as stated.

We will prove this to be true with a proof by induction.

Let i=0. This means we need to show that the statement "If Dijkstra's is run on a graph with no negative edge weights: at the start of the 0th (first) iteration of the while loop, if u is the closest unprocessed vertex, u.dist stores the shortest path distance from source to u" holds true.

Before the first iteration of the while loop, all vertices are marked as unprocessed, the source vertex is set to a distance of 0, and all other vertices are set to distance infinity. We then enter the while loop, and the closest unprocessed vertex is the source vertex and the distance to it is 0. Since we assumed there are no negative edge weights in the graph, it is impossible at any point for the distance to the source to ever be less than 0. Additionally, the vertex is marked as processed at the end of this iteration, and we never modify its distance during the rest of this iteration, so we never modify its distance again during the course of the algorithm. Therefore, 0 is the current stored distance and is the shortest path distance from source to itself. The statement therefore holds for i=0 by inductive hypothesis.

Let us assume that the statement is true for some integer k. If we can prove that it is true for k+1, then we know that it is true for all $i >= 0$. Let us also assume that k+1 <= the number of total iterations through the while loop, so we are guaranteed to enter the while loop.

Let i=k+1. This means we need to show that the statement "If Dijkstra's is run on a graph with no negative edge weights and at the start of the kth iteration of the while loop, the closest unprocessed vertex stores the shortest path distance from source to the vertex, at the start of the k+1th iteration of the while loop, if u is the closest unprocessed vertex, u.dist stores the shortest path distance from source to u" holds true.

If we enter the while loop on the k+1th iteration, we know that there is at least 1 vertex left unprocessed. Logically following the flow of Dijkstra's algorithm, the closest unprocessed vertex must connect to one of the previously processed vertices as there is no other way to reach it. Since we know that the statement is true for i=k, that means the shortest distance to all previously processed vertices are known, as k iterations through the while loop mean that k vertices have been processed and we know those distances. During the k+1th iteration, we find the closest unprocessed vertex, and since we know that this connects to one of the previously processed vertices, we have an associated distance with it. The vertex is marked as processed at the end of this iteration, and we never modify its distance during the rest of this iteration, so we never modify its distance again during the course of the algorithm. Therefore, the final distance for this unprocessed vertex is the distance known at the beginning of the iteration. The statement therefore holds for k+1, and we have proven by inductive hypothesis that the statement is always true.

By the principle of induction, this statement holds true for all i¿=0.

Question 4

a) “If Dijkstra’s algorithm does not return the shortest paths of G from the chosen source vertex, then G has negative edge weights.”

b) “If Dijkstra’s algorithm returns the shortest paths of G from the chosen source vertex, then G has no negative edge weights.”

c) I could not find a way to insert my image in so I will describe my graph instead: There are 5 nodes (A, B, C, D, E) on an undirected, weighted graph and the source vertex is A. The edges in the graph are: (A, B): 1, (B, C): 5, (A, D): 3, (D, E): -10. The converse is false in this case because Dijkstra’s correctly identifies the shortest distance to all nodes as (A: 0, B: 1, C: 6, D: 3, E: -7). This is true despite the fact that there are negative edge weights, so the converse is false.

Question 5

Statement: You are one of the n riders being matched to n horses. You are one of k popular riders. Every horse has the k popular riders as their k top choices (though they might not agree on the ordering of the k riders). If Gale-Shapley is run with riders proposing, you will be matched to one of your top k choices.

We will prove this statement by contradiction by proving the following statement to be false: You are one of the n riders being matched to n horses. You are one of k popular riders. Every horse has the k popular riders as their k top choices (though they might not agree on the ordering of the k riders). If Gale-Shapley is run with riders proposing, you may not be matched to one of your top k choices.

If you are not matched with one of your k top choices, that means all k of them are matched to another rider that they rank higher than you. This is because we know that if any horse was matched with someone they rank lower than you, that would form a blocking pair since you are either not matched with anyone or are matched with someone you rank lower than those k horses. Since all horses agree on the top k riders, including you, this means that k horses have been perfectly matched to their top k-1 riders and an additional rider. We know there is an additional rider because if k horses were matched to k-1 riders, then we would not have perfect matching, which is a guarantee of Gale-Shapley algorithm. This number excludes you since you are not matched to any of these k horses. However, the horses cannot be matched to this additional rider because you have ranked all k horses higher than any other horse you could be matched with, and they have ranked you higher than the additional rider. Therefore, it is impossible for you to not be matched to one of your top k choices, meaning you will be matched to one of them.

Question 6

Yes, it is possible for no agent to receive their first choice. An example for this would be for the set of riders, R, containing A, B, C, D, and the set of horses, H, containing a, b, c, d (both sets have 4 elements). If the order of preferences for the horses and riders are the following (horse's preference, rider's preference):

Agents	A	B	C	D
a	(1,3)	(2,3)	(3,2)	(4,3)
b	(1,4)	(4,1)	(3,3)	(2,2)
c	(2,2)	(1,4)	(3,4)	(4,1)
d	(4,1)	(2,2)	(3,1)	(1,4)

Then we follow Gale-Shapley's algorithm with the rider's proposing:

(A, d) are paired because both are free.

(B, b) are paired because both are free.

(C, d) are paired because d is C's first choice and d prefers C over A, d's current matching. Therefore, A is matched with c because it is A's next choice and c is free, so (A, c) are paired.

D and c are not paired because although c is D's first choice, c is already matched with A, who c prefers over D. (D, b) are paired because b is D's first remaining choice and although (B, b) are currently a pair, b prefers D over B. Now that B is free, B is matched with d because d is B's first remaining choice and although (C, d) are currently paired, d prefers B over C. Now that C is free, it is matched with a as it is C's second choice and currently free.

Our final pairings are now (A, c), (B, d), (C, a), (D, b). This is a stable matching because there are no blocking pairs and there is perfect matching. A was paired with their second choice, B with their second choice, C with their second choice, and D with their second choice. a was paired with their third choice, b with their second choice, c with their second choice, and d with their second choice. Therefore, no agent received their first choice in this example, so it is possible for no agent to receive their first choice.

Question 7

If we use our standard Gale-Shapley algorithm, we end up with proposer-optimality where either the riders or the horses have proposer-optimality. Let us assume that riders are the ones proposing and that all riders/horses have ranked every member of the opposite set. We will use c=1.2.

For the first rider, r_1, their order of preference will be $h_1, h_2, ...h_n$. For all riders afterwards from 2 to n, their order of preference will be $h_i, h_1...h_n$, where $h_1...h_n$ is in increasing order from 1 to n with h_i just removed from the ordering. For example, for i=2 this is $h_2, h_1, h_3...h_n$ and for i=3 this is $h_3, h_1, h_2, h_4...h_n$.

For all horses from $1 <= i <= n-1$, their order of preference is $r_{i+1}, r_{i+2}...r_n, r_i, r_{i-1}, ...r_1$. For example, for i=1 this is $r_2, r_3...r_n, r_1$ and for i=2 this is $r_3, r_4, r_n...r_1, r_2$. For horse n, their order of preference will be $r_1, r_2, ...r_n$.

The maximum amount of stable-matches that can occur can be seen with the following: There is one matching that is completely rider-optimal and one matching that is completely horse-optimal (2 total). If n=2, there are only these 2 pairings. If n=3, there are 3 more pairings as we can pair any rider r_i with its first choice, and then whichever two horses are unpaired can be paired with their top remaining choices. This can be repeated for i=1, 2, and 3. Attempting this with one horse having its first choice results in duplicate matches, so we have 5 total pairings. If n=4, we can repeat a similar strategy but also try having 2 riders get their first choices. This results in a total of 12 stable matches. For n=5, we can repeat the strategies for n=4 and n=3, but also try a mix of 3 rider-optimal matches and 2 horse-optimal matches, or vice versa. This results in a total of 29 stable matches. If T(n) represents the total number of stable matches possible for n, then T(5)=T(3)+2T(4), or T(n)=T((n/2)+1)+2T(n-1) where n/2 rounds down to the nearest integer. We can test this again with a larger integer, let's say n=9. This should be equal to T(5)+2T(8). We know T(5) = 29. 2T(8) by the same formula = 2T(5)+4T(7) = 58 + 4T(7). 4T(7) = 4T(4)+8T(6). 8T(6) = 8T(4)+16T(5) = 1448. 4T(7) = 48 + 1448 = 1496. T(8) = 58 + 1496 = 1554. T(9) = 1583 = 29 + 1554. Therefore, T(n)=T((n+2)/2)+2T(n-1):

The first part of this runs in log(n) time. The second part can be calculated by tree method:

Level	Nodes per Level	Work per Node	Total Work at Level
0	1	1	1
1	2	1	2
2	4	1	4
3	8	1	8
...	...	...	...
n	2^n	1	2^n

Input size at level i: n-i

Work per node at level i: 1

Number of nodes at level i: 2^i

Total work at level i: 2^i

Last level of i: n

Total work at last level: 2^n

T(n) = $\sum_{i=0}^{n-1} 2^i + 2^n$

T(n) $\in \theta(2^n)$. Therefore, the number of stable matches is $\in \Omega(2^n)$

Question 8

(a) We can modify the standard stable matching algorithm so that it accounts for unacceptable pairings and potentially mismatched job pairings and job applicants:

Initialize the set of jobs J and set of applicants A. When initializing these sets, we only rank the preferences of those that are not considered unacceptable. In other words, the preference list does not contain any elements of the other set that are found unacceptable. Next, find whichever set has less elements. Out of sets J and A, we will refer to whichever happens to be the smaller set as N. If both sets are the same size then either can be used as N. Set the Gale-Shapley algorithm to use the set N as the proposers.

While there exists an unmatched element (n) in N who still has an element (m) in their list of preferences that they have not already tried matching with:

i) Find m, the highest ranking element among their preferences in the other set that n has not tried matching with yet.

ii) If m is unmatched and n is in m's list of preferences then match (n, m).

iii) If m is already paired with some element (n2), check if m prefers n over n2. Pair m with whichever between n and n2 that m ranked higher. If this is n, then unmatch m and n2 so n2 is free to find a new match afterwards. Add the unmatched element to the queue of elements that need to still be paired.

(b) To prove that our algorithm results in a stable matching, we will prove that the final pairings are all stable with a proof by contradiction. We will assume that J is the smaller set, so j gets the advantage of proposer-optimality, but the proof applies regardless of if J or A chooses (just switch the two).

Let us assume that there exists some j in J that is paired with some a in A and the two form a blocking pair. If this is the case, then there are two cases: there exists some free element in A that j prefers to a or there exists some free element in J that a prefers to j.

Case 1: If there exists some element (a2) in A that j prefers over a, then by our algorithm, if a2 is either free or prefers j over its current pairing, then j and a2 would be paired. The reason they are not paired is because a2 is already paired with an element in J that a2 prefers over j, or a2 finds j unacceptable and has not listed them on their list of preferences. This means that j and a do not actually form a blocking pair. Therefore, there must not be a blocking pair, meaning the assignment is stable.

Case 2: If there exists some element (j2) in J that a prefers over j, then by our algorithm, a and j2 would be paired unless j2 was already paired with an element in A that they preferred over a. If j2 was free or ranked a higher than its current partner, then they would be paired together. Since a and j2 did not form a pair, j2 must have its current pairing ranked higher than a, or finds a unacceptable and has not listed them on their list of preferences. Therefore, there must not be a blocking pair, meaning the assignment is stable.

Since there are no blocking pairs in every case, we can be sure that our output produces a stable assignment.

(c) To initialize the sets based on the inputs, the runtime is the same as the standard Gale-Shapley algorithm ($\theta(n^2)$). Identifying the smaller set can be done in worst case O(n) runtime and generating each preference list can be done in n^2 runtime as well. The initial set up of Gale-Shapley is unchanged, so it will still take $O(n^2)$ runtime.

If given a list of n horses and n riders, our while loop will run in $\theta(n^2)$ time because nobody will request a pairing with a member of the other set more than once, so there will be a maximum of n^2 pairings and we can perform each pairing in constant time. Therefore, this algorithm runs in $\theta(n^2)$ time as every part runs in n^2 time.

Question 9

(a) I will be implementing a small reduction of Gale-Shapley's algorithm. Due to the extreme matchings portion of this question, I will have to run the algorithm with either the horses as proposers or riders as proposers depending on the situation. I will therefore have to use temporary variables that I decide the value of depending on which group is the proposer. I will also have to modify the output array to ensure that it always uses horses as the indices and riders as the pairings of each horse regardless of who is the proposer.