James Sweetman (jts2939)
Jung Yoon (jey283)
Unique #51835

Assignment 1

Q1(a)

f6, f7, f4, f3, f5, f1, f2

Q1(b)

A. In the case of Alice's Algorithm, the algorithm is incorrect and we can prove this by a counterexample. Let's assume that the input stream consists of a singular integer value, i = 10. Then, given the input, 10 is added to the sum which becomes 10. Then, the average which is sum / count = 10 / 0 is reported as the average. Then, the count is increased by 1 so its value is 1. Note that the reported average is undefined. This is because the count was not increased prior to the average being calculated and reported. To fix the problem, either the step "Increase count by 1" should be moved to be before the "Report average as sum / count" step OR the "Report average as sum / count" should be modified to be "Report average as sum / (count + 1)". Either modifications would rectify the error in the count and would make the algorithm correct.

B. In the case of Bob's Algorithm, the algorithm is correct and we can prove it by induction.

To prove that the algorithm is correct, we must assume that the current average before entering the loop again is correct. And to prove that this calculates the average, we show that after going through the loop k times, P(a1+...+an) = a1+...+an / k. This is a loop invariant.

→ Basis step: Let's assume that the first value in the input stream is an integer A1 = 1. Before entering the loop the current average and count, K, are both 0. The average is modified to be 1 since P(A1) = P(1) = (0*0+1) / (0+1) = 1. The average is correct. Then, the count is incremented to 1 and we know that K is 1 in this case so it works. Thus, the basis step is proven.

→ Induction step:

For an arbitrary integer in the stream, we know the current average and the count which should be K-1 at the start of the loop.

Inside the loop, note that the integer is read and that the the algorithm to calculate the new_average is simply averaging it with the previously calculated average. Since we

can assume that the current average (the average calculated in the directly previous loop) is correct, we know that the new_average is correct. The algorithm multiplies the current average with K-1 and adds it to the integer and divides it by K, meaning that it averages the current and new averages. Also, the count is increased by one so we also know that the count is correct since then, count = k at the end of the loop.

Since both of the conditions are met (reported average is correct and count is correct) at the end of each arbitrary loop, we know that the algorithm is correct by induction.

End of proof.

Q1(c)

Picture:

Proof by Induction:

Base Case:

$$\sum_{i=0}^{a-1} a^i = \sum_{i=0}^{1-1} a^i = a^0 = 1 = \frac{1-a^1}{1-a} \quad \checkmark$$

Inductive Hypothesis:

Assume $p(n)$ is true such that

$$\sum_{i=0}^{n-1} a^i = \frac{1-a^n}{1-a}$$

Prove: $\sum_{i=0}^{(n+1)-1} a^i = \frac{1-a^{(n+1)}}{1-a}$

$$\begin{aligned}
\text{RHS} &= \sum_{i=0}^{n} a^i \\
&= \sum_{i=0}^{n-1} a^i + a^n \\
&= a^n + \frac{1-a^n}{1-a} \\
&= \frac{(1-a)a^n}{1-a} + \frac{1-a^n}{1-a} \\
&= \frac{(1-a)a^n + 1 - a^n}{1-a} \\
&= \frac{a^n - a^{n+1} + 1 - a^n}{1-a} \\
&= \frac{1-a^{n+1}}{1-a} \\
&= \text{RHS} \quad \checkmark
\end{aligned}$$

Q2(a)

This is just binary an implementation of the binary search algorithm. The worst case scenario would be O (log n) which means that O (log (1,000,000)) = 19.931569. In the worst case scenario, we would need 20 questions to find the correct number. To be more specific:

WORST CASE SCENARIO (Given that 2 is the number to guess)
1,000,000
500,000 --- 1st
250,000 or 750,000 --- 2nd
125,000 --- 3rd
62,500 --- 4th
31,250 --- 5th
15,625 --- 6th
7812 --- 7th
3906 --- 8th
1953 --- 9th
976 --- 10th
488 --- 11th
244 -- 12th
122 -- 13th
61 --- 14th
30 --- 15th
15 --- 16th
7 --- 17th
3 --- 18th
1 --- 19th
2 --- 20th

Q2(b)

Claim: Given that you must guess a number between 1 and N and 3 strikes, then it is possible to correct guess the said number using O(N^1/3).

Proof:
First, we can partition the range by N^(2/3) such that the ranges would be multiples of N^(2/3) → → → i.e. N^(2/3), 2*N^(2/3), 3*N^(2/3), …, N-N^(2/3). Then, we would start guessing from the first partition, asking if the number is greater than the partition (i.e. “is the number greater than N^(2/3)”, “is the number greater than 2*N^(2/3), etc.) until we reach our first “no”.

After we reach our first “no”, we should know which N^(2/3) partition that our number belongs in since we know it’s greater than the partition just before the “no” (we will call it P1 from hereon) and less than the partition that we got the “no” for.

Then, we split that N^(2/3) partition further into ((N^(2/3))^1/3) and start guessing from the smallest partition first, asking if the number is greater than the partition (i.e. “is the number greater than (P1 + (N^(2/3))^1/3)”, “is the number greater than P1 + 2*((N^(2/3))^1/3)”, etc.) until we are given our second “no”.

Note that as before, we will label the partition reached just before the second “no” as P2, of which we know that our number is greater than. Also note that the partition that we received our second “no” for is greater than the number we need to guess.

Then, since we have run out of all but 1 strike, we can no longer take any risks with our guesses. Therefore, we need to stop partitioning and just go on asking from P2 and ask if the number we want is greater than each consecutive number until we reach our last and final “no”.

Once given a no, we can guess that the number that we got a “no” for is the correct answer.

Note that since we partitioned each time by multiples of 1/3, we can ensure that it is possible to correctly guess the number using O(N (1/3)) guesses.

End of proof.

Q3

→ Modifications: for the most part, the Gale-Shapely algorithm proves sufficient. However, given that the “men”, or the committees in this case, can request 0 or more “women”, the interns, we must modify the algorithm such that each “man”/committee has the possibility of being in a polygamous relationship. More specifically, we must make it so that you fill each spot in each consecutive committee before moving on to another committee. In any other aspect, the Gale-Shapely algorithm is perfectly fine.

→ Proof of termination:
Claim: Algorithm terminates after at most n^3 iterations of a nested while loop.
Proof: Each time through the while loop a committee proposes to a new woman for how many slots it has. There are only N^3.

→ Proof of stability: (Contradiction)
Claim: The modified G-S algorithm is stable, i.e. there are no unstable pairs.
Proof by contradiction:
Assume that an unstable pair, Team X- Applicant Y, did occur.

Instability #1:

Assume Applicant X and Team Y prefer each other over their current matches. This means that within the matchings of the G-S algorithm used to find a matching for each spot, an instability was made. This is a contradiction to the fact that the G-S algorithm returns only valid partners.

Instability #2:
Case 1:
Assume there is a Team Y and Applicant X, where X is not on a team and Y has X' on their full team while X > X' in their rankings. This means that Team Y proposed to X' before X. This contradicts the assumption where X > X' for Team Y.

Case 2:
Assume Team Y has Applicant X' and an opening on their team, while Applicant X is not assigned to any team and has been ranked X > X' by Team Y. Since the G-S algorithm is applied to every spot open on a team, this means that every applicant left has to have a team. However, this contradicts the assumption where Applicant X does not have a team.

Instability #3:
Assume Applicant X is on Team Y' when Team Y has an opening and X prefers Y to Y'. This means that X was never proposed to by Y. This contradicts our algorithm because the G-S algorithm is applied to every open spot, meaning that Y would have to propose to X eventually and since X prefers Y to Y', X would swap to move up in their rankings.

End of proof.