Name: Luna McBride
ID: 107607144

CSCI 3104, Algorithms **Profs. Hoenigman & Agrawal**
Problem Set 1a (10 points) **Fall 2019, CU-Boulder**

Advice 1: For every problem in this class, you must justify your answer: show how you arrived at it and why it is correct. If there are assumptions you need to make along the way, state those clearly.

Advice 2: Verbal reasoning is typically insufficient for full credit. Instead, write a logical argument, in the style of a mathematical proof.

Instructions for submitting your solution:

- The solutions **should be typed** and we cannot accept hand-written solutions. Here's a short intro to Latex.
- You should submit your work through **Gradescope** only.
- If you don't have an account on it, sign up for one using your CU email. You should have gotten an email to sign up. If your name based CU email doesn't work, try the identikey@colorado.edu version.
- Gradescope will only accept **.pdf** files (except for code files that should be submitted separately on Gradescope if a problem set has them) and **try to fit your work in the box provided**.
- You cannot submit a pdf which has less pages than what we provided you as Gradescope won't allow it.

Name: Luna McBride
ID: 107607144

CSCI 3104, Algorithms **Profs. Hoenigman & Agrawal**
Problem Set 1a (10 points) **Fall 2019, CU-Boulder**

1. *(3 pts) What are the three components of a loop invariant proof? Write a one-sentence description for each one.*

   1) Initialization; This is the step where you make sure the invariant holds before even entering the loop.
   2) Maintenance; This is where you check if the invariance holds for each individual loop. It does not matter if you check immediately after the last loop or immediately before the next, but it must be checked to make sure this applies for all values.
   3) Termination; At this point the beginning and middle has been checked, however, it must also end by following the invariance to truly fit. Once we have the end value, that makes all values checked and we can then definitively prove the relation.

2. *(6 pts total) Identify the loop invariant in the following algorithms.*

   (a)
   ```
   FindMaxElement(A) : //suppose array A is not empty
           ret = A[0]
           for i = 1 to length(A)-1 {
               if A[i] > ret{
                   ret = A[i]
           }}
           return ret
   ```

Name: Luna McBride
ID: 107607144

CSCI 3104, Algorithms **Profs. Hoenigman & Agrawal**
Problem Set 1a (10 points) **Fall 2019, CU-Boulder**

Well, a maximum is the highest value, so the invariance should be that a value is the highest up until now.
Invariance: (A[0],...,A[i-1]) $\leq$ ret for $0 \leq$ i $\leq$ length(A)-1 (deincremented as the for loop increments automatically)
Initialization: We start with ret=A[0], which is the highest of all the values we have looked at thus far, being itself.
Maintenance: The if statement takes sets the ret to the new highest if A[i] (before increment) is bigger, forcing it to be the highest of the values seen such far. That means all others could be less than or equal to the current highest ret, but never greater as the highest was just taken by the if statement. Termination: The final value (length(A)-1 value, as we start at 0), once checked either takes the last highest value or holds a previous value that is highest for ret. There is no chance of a value greater (if coded it right, that is), and thus is the ret value can only be greater than or equal to absolutely everything else in array A.

(b)
```
FindElement(A, n) : //suppose no duplicates in array A and array A is not empty
    ret = -1 //index -1 implies the element haven't been found yet
    for i = 0 to length(A)-1 {
        if A[i] == n{
            ret = i
    }}
    return ret
```

Note: I did parts a and c before checking piazza and thus seeing to just type the invariance, but I do not want to delete all that typing and it is how I thought it out. sorry.
Invariance: ret $\geq$ -1

(c)
```
SumArray(A) : //suppose array A is not empty
    sum = 0
    for i = 0 to length(A)-1 {
        sum += A[i]
    }
    return sum
```

Invariance: sum=A[0]+A[1]...+A[i-1] for 0 $\leq$ i $\leq$ length(A)-1 (Assuming a +0 for the base, however, that is unnecessary)
Initialization: sum=0, which takes the case of the assumed 0, not including the A[i-1] as -1 is outside the range and would probably break the computer
Maintenance: Sum adds the array value at each value i in the loop. Following the idea listed in C++ logic (i incremented at the end of the loop as part of the loop deinition (i=0;i¡x;i++)), the i needs to be deincremented to get the correct index when before or after an iteration to count it.
Termination: the last value is at the length of A (-1 as to account for the index start at 0). The i still increments, but the for loop stops it after the increment (the i¡x in (i=0;i¡x;i++)), making the last i value length(A) and the last added value A[i-1], the total amount of indecies to account the whole array.

Name: Luna McBride
ID: 107607144

CSCI 3104, Algorithms **Profs. Hoenigman & Agrawal**
Problem Set 1a (10 points) **Fall 2019, CU-Boulder**

3. *(1 pt) If r is a real number not equal to 1, then for every $n \geq 0$,*

$$\sum_{i=0}^{n} r^i = \frac{(1 - r^{n+1})}{(1 - r)}.$$

Provide the first two steps of a proof by induction i.e. base case and the inductive hypothesis. You will be asked to complete this proof later in ***PS1b****.*

Base case: i=0
$\sum_{i=0}^{0} r^i = \frac{(1-r^{n+1})}{(1-r)}$
$r^0 = 1$
$\frac{(1-r^{0+1})}{(1-r)}$
$\frac{(1-r)}{(1-r)}$
$\frac{1-r}{1-r} = 1$

$r^0 = \frac{(1-r^{0+1})}{(1-r)}$
$1 = 1$ (Plugged in equivalent values shown above)
Therefore, since 1 does equal 1, the equation holds for the base case

Inductive Hypothesis: If the equation works for the base case, it should be fair to assume this should work for some integer k. $\sum_{i=0}^{k} r^i = \frac{(1-r^{k+1})}{(1-r)}$ where k $\geq$ 0 and r != 1