Name: Alexander Hawkins
ID: 105070938

CSCI 3104, Algorithms **Charlie Carlson & Ewan Davies**
Problem Set 1 – Due Setpember 4th **Fall 2020, 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: Informal reasoning is typically insufficient for full credit. Instead, write a logical argument, in the style of a mathematical proof.

Instructions for submitting your solutions:

- The solutions **should be typed using** LaTeXand we cannot accept hand-written solutions. Here's a short intro to LaTeX.
- You should submit your work through the **class Canvas page** only.
- You may not need a full page for your solutions; pagebreaks are there to help Gradescope automatically find where each problem is. Even if you do not attempt every problem, please submit a document with at least as many pages as the blank template (or Gradescope has issues with it). **We will not accept submissions with fewer pages than the blank template**. Submissions with more pages are fine.
- **You must CITE any outside sources you use, including websites and other people with whom you have collaborated. You do not need to cite a CA, TA, or course instructor.**
- **Posting questions to message boards or tutoring services including, but not limited to, Chegg, StackExchange, etc., is STRICTLY PROHIBITED. Doing so is a violation of the Honor Code.**

Quicklinks: 1 2 3 4

Name: Alexander Hawkins
ID: 105070938

CSCI 3104, Algorithms **Charlie Carlson & Ewan Davies**
Problem Set 1 – Due Setpember 4th **Fall 2020, CU-Boulder**

Problem 1. Prove by induction that for each $n \in \mathbb{Z}^+$,

$$\sum_{k=1}^{n} \frac{1}{k^2} \leq 2 - \frac{1}{n}.$$

Proof. □

Statement of Induction:

I am showing

$$\sum_{k=1}^{n} \frac{1}{k^2} \leq 2 - \frac{1}{n}$$

for all positive integers n via induction on n.

Base Case:

If n = 1 (smallest element)

$$\sum_{k=1}^{1} \frac{1}{1^2} \leq 2 - \frac{1}{1}$$

$$\sum_{k=1}^{1} 1 \leq 1$$

Both sides of the summation are equivalent to each other and create the correct output for n = 1 as desired.

Statement Inductive Hypothesis:

Let h = n be a positive integer such that

$$\sum_{k=1}^{h} \frac{1}{k^2} \leq 2 - \frac{1}{h} \quad h \in \mathbb{Z}^+$$

holds true for a positive integer h.

Inductive Step:

Let h = n + 1 such that

$$\sum_{k=1}^{h+1} \frac{1}{k^2} \leq 2 - \frac{1}{h+1} \quad h \in \mathbb{Z}^+$$

remains true for a positive integer h.

$$\sum_{k=1}^{h+1} \frac{1}{k^2} \leq (\sum_{k=1}^{h} \frac{1}{k^2}) + \frac{1}{(h+1)^2}$$

$$\sum_{k=1}^{h+1} \frac{1}{k^2} \leq 2 - \frac{1}{h} + \frac{1}{(h+1)^2} \quad \textbf{By I.H.}$$

$$\sum_{k=1}^{h+1} \frac{1}{k^2} \leq 2 + \frac{h-(h+1)^2}{h(h+1)^2}$$

$$\sum_{k=1}^{h+1} \frac{1}{k^2} \leq 2 + \frac{-h^2-h-1}{h(h+1)^2}$$

$$\sum_{k=1}^{h+1} \frac{1}{k^2} \leq 2 - \frac{h^2+h}{h(h+1)^2} - \frac{1}{h(h+1)^2}$$

$$\sum_{k=1}^{h+1} \frac{1}{k^2} \leq 2 - \frac{1}{h+1} - \frac{1}{h(h+1)^2}$$

$$\sum_{k=1}^{h+1} \frac{1}{k^2} \leq 2 - \frac{1}{h+1} - \frac{1}{h(h+1)^2}$$

From calculations,

$$\sum_{k=1}^{h+1} \frac{1}{k^2} \leq 2 - \frac{1}{h+1} - \frac{1}{h(h+1)^2} \leq 2 - \frac{1}{h+1}$$

for all positive integers h. In conclusion, the induction holds true for all positive integers.

Problem 2. What are the three components of a loop invariant proof? Write a 1–2-sentence description for each one.

The three components of a loop invariant proof include:

- Initialization
  - It is true prior to the first iteration of the loop.
- Maintenance
  - If it is true before an iteration of the loop, it remains true before the next iteration.
- Termination
  - When the loop terminates, the invariant gives me a useful property that helps show the correctness of the algorithm (expected outcome).

Problem 3. Consider the following algorithm.

```
FindMinElement(A[1, ..., n]) : //array A is not empty
    ret = A[n]
    for i = 1 to n-1 {
        if A[n-i] < ret{
            ret = A[n-i]
    }}
    return ret
```

Do the following.

(a) Suppose a student provides the following: *At the start of each iteration, i is one more than the number of iterations that have occurred.* Is this a valid loop invariant? Justify your answer in light of Problem 2.

Answer:

Yes, this is a valid loop invariant because the initialization is true prior to the first iteration of the loop which i is set to 1. The variable i being set to 1 creates i being one more than the number of iterations that have occurred. It is true before an iteration of the loop begins which it remains true before the next iteration, after every iteration, and at the end of the iteration.

(b) Is the above invariant in part (a) *useful* in proving that the FindMinElement algorithm is correct? If so, explain why. If not, give a *useful* loop invariant and explain why your invariant is useful in proving the algorithm correct. **Note that this question is *not* asking you to prove that the algorithm is correct.**

Answer:

No, the above invariant in part(a) is not *useful* in providing that the FindMinElement algorithm is correct. The correctness of the algorithm is not evident. If *ret* happened to be less than the set of elements traversed through the array, then there would be a useful loop invariant such that A[h] has index h = (n - i). The given loop invariant is true prior to the first iteration of the loop, maintains true before the next iteration, and provide us with the minimum element in the termination process.

Name: Alexander Hawkins
ID: 105070938

CSCI 3104, Algorithms **Charlie Carlson & Ewan Davies**
Problem Set 1 – Due Setpember 4th **Fall 2020, CU-Boulder**

Problem 4. Consider the following algorithm. We seek to prove that the algorithm is correct using a loop invariant proof.

```
ProductArray(A[1, ..., n]) : //array A is not empty
    product = 1
    for i = 1 to n {
        product = product * A[i]
    }
    return product
```

(a) Provide a loop invariant that is *useful* in proving the algorithm is correct.

A[1] = A[1]

A[2] = A[1] * A[2]

A[3] = A[1] * A[2] * A[3]

A[4] = A[1] * A[2] * A[3] * A[4]

A[5] = A[1] * A[2] * A[3] * A[4] * A[5]

product = A[i] * A[i + 1]

when i = 1, product = 1 * A[0] (1st index)

when i = 2, product = 1 * A[0] * A[1]

when i = 3, product = 1 * A[0] * A[1] * A[2]

product = 1 * A[i - 1] * A[i - 2] * A[i - 3] ...

Following the steps of the algorithm, the useful loop invariant is correct. Product contains the product in the sub array A[1, ..., i] that matches the return value.

(b) Using the loop invariant above, provide the **initialization** component of the loop invariant proof. That is, show that the loop invariant holds before the first iteration of the loop is entered.

Proof.

The loop invariant holds true prior to the first iteration because prior to the loop beginning, the variable i = 1 sets the array to A[i - 1] = 1 * A[0] which is the elements of the first product iteration. In this case, A[i] begins as an empty array. Therefore, the empty array multiplied with the product, is the product. In result, the loop invariant holds true prior to the first iteration of the loop. □

(c) Using the loop invariant above, provide the **maintenance** component of the loop invariant proof. That is, assume the loop invariant holds just before the i-th iteration of the loop, and use this assumption to show that it still holds just before the $(i+1)$-st iteration.

Proof.

If the invariant was previously true and the sub-array A[1, ... , i] has not reached n (which ends the iterations), product is not yet updated and the invariant remains true. Assuming the loop invariant holds true prior to the i-th iteration, our product is 1 * A[i - 1] * A[i - 2] * A[i - 3] * ... through the last element (of the array) which is 1 * A[0] * A[1] * A[2] * ... through the last element (of the array). Therefore, the product of the (i+1)-st iteration will be 1 * A[(i+1) - 1] * A[(i+1) - 2] * A[(i+1) - 3] * ... through the last element (of the array) which is 1 * A[1] * A[2] * ... through the last element (of the array) which still holds true just before the (i+1)-st iteration. □

(d) Using the loop invariant above, provide the **termination** component of the loop invariant proof. That is, assume the loop invariant holds just before the last iteration. Then argue that the loop invariant holds after the loop terminates, based on what happens in the last iteration of the loop. Finally, use this to argue that the algorithm overall is correct.

Proof.

We can assume the iteration holds true prior to the ith iteration such that our product = 1 * A[i - 1] * A[i - 2] ... through our completion of iterations. After all iterations are complete, the sub-array in the invariant is the product of entire array of A[1, ..., n], and the algorithm will return as claimed. When the algorithm returns through completion, the invariant gives me a useful property of the finished product which shows the correctness of the algorithm from the expected outcome. □