

Homework 1

Dathrika Saicharitha

02010404

1.b. Before other steps in the proof of loop invariant, we need a loop invariant.

In the algorithm we have a variable answer, which can be varied while the loop is running. This means we are incrementing and forming an answer. Here we used for loop to form a solution incrementally.

For the above algorithm, the first element in the output array R is taken same as the first element of the input array A because of the if condition.

For loop is used to count from 2 to highest index, then we can process each element of an array.

At the beginning of each iteration of the for loop, indexed by j, the subarray consists of R [1.... j-1] consists cumulative sums of an array A

We state these properties of R [1...j-1] as loop invariant.

1.c. We must show three things about the loop invariant to prove the correctness of the algorithm. They are initialization, maintenance and termination.

Initialization:

It is showing the loop invariant holds true before the first loop iteration, when $j=2$. The subarray R [1...j-1], therefore, consists of just the single element R [1], which is the original element in A [1]. Also, this subarray consists of expected output array elements i.e., cumulative sums of input array A, which shows that the loop invariant holds true prior to the first iteration of the loop.

Maintenance:

It is showing that each iteration maintains the loop invariant.

The loop invariant holds true at the start of the iteration j.

Then it must be that answer, which contains the cumulative sums of input array elements A.

First, input array A has length of n

let $n=4$ and $A[4] \Rightarrow [3, 2, 8, 5]$

then the output array R also has the length of n.

$$R[1] = A[1] = 3$$

Next, at the start of iteration $j=2$,

$$R[2] = R[1] + A[2] = 3+2 = 5$$

At the start of iteration $j+1=3$,

$$R[3] = R[2] + A[3] = 5+8 = 13$$

When $j=4$

$$R[4] = R[3] + A[4] = 13+5 = 18$$

When j reaches $\text{length}(A) \Rightarrow n$ loop terminates. So we now have the output array R as follows:

$R = [3, 5, 13, 18]$ which is cumulative sum of input array elements.

So incrementing j for the next iteration of the for loop preserves the loop invariant.

1.a. Termination:

We formulated a loop invariant and after going out of the loop, gives us the statement with which we can prove correctness.

The condition causing For loop to terminate is $j > \text{length}(A) \Rightarrow j > n$ where $n = \text{length}(A)$

Each loop iteration increases j by 1, and when $j=n$, the loop invariant gives the cumulative sum of numbers in sub array $R[0: n]$. This is exactly the value that the algorithm should output which it then outputs. Thus, the algorithm is correct.

1.d. Consider each statement takes a unit of time

then run time is evaluated as follows:

	<u>run time</u>
$R \leftarrow \text{new array with length}[A] \text{ elements}$	- 1
if $\text{length}[A] > 0$	- 1
$R[1] \leftarrow A[1]$	- 1
end if	
for $j \leftarrow 2$ to $\text{length}[A]$	- n
$R[j] \leftarrow R[j-1] + A[j]$	- n-1

end for

return R

- 1

$$f(n) = n + (n-1) + 4$$

$$= 2n + 3$$

Time complexity $T(n) = O(n)$

2. $f = O(g)$ and $g = O(h)$

By Big Oh definition there exist positive constants 'd' and n_0 such that $0 \leq f(n) \leq d \cdot g(n)$ for all $n \geq n_0$

$$f(n) \leq d_1 \cdot g(n)$$

$$g(n) \leq d_2 \cdot h(n)$$

By substituting $g(n)$ in $f(n)$, we get

$$f(n) \leq d_1 \cdot (d_2 \cdot h(n))$$

Where constants $d_1 \cdot d_2 = d$, then

$$f(n) \leq d \cdot h(n)$$

$$f(n) \leq O(h(n))$$

Hence, if $f(n) = O(g(n))$ and $g(n) = O(h(n))$ then $f(n) = O(h(n))$

3. a. $f(k) = O(g(k))$ implies $g(k) = O(f(k))$

This statement is false, we can disprove as below:

By Big Oh definition there exist positive constants 'd' and k_0 such that $0 \leq f(k) \leq d \cdot g(k)$ for all $k \geq k_0$

Let $f(k) = k^2$ and $g(k) = k^3$

Then $k^2 = O(k^3)$ is true, because we always assume that $g(k)$ grows faster than $f(k)$. for example, $d=1$ then $k^2 \leq k^3$ for every $k \geq 1$

But the reverse is not true i.e., k^3 cannot be $O(k^2)$. Because this means $k^3 \leq d \cdot k^2$ for every $k^3 - d \cdot k^2 \leq 0$ but this is false.

3. b. $f(k) + (g(k)) = \theta(\min(f(k), g(k)))$

This statement is false, we can disprove as below:

Let $f(k) = k^3$ and $g(k) = k^2$, for constant $=1$

$$f(k) + (g(k)) = k^3 + k^2$$

$$\theta(\min(f(k), g(k))) = \theta(\min(k^3, k^2)) = \theta(k^2)$$

$$k^3 + k^2 \neq \theta(k^2)$$

3.c. $f(k) = O(g(k))$ implies $\lg(f(k)) = O(\lg(g(k)))$, where $\lg(g(k)) \geq 1$ and $f(k) \geq 1$ for all sufficiently large k

This is true

$f(k) = O(g(k))$ such that $0 \leq f(k) \leq d \cdot g(k)$ for all $k \geq k_0$

$$0 \leq \lg(f(k)) \leq \lg(d \cdot g(k))$$

$$0 \leq \lg(f(k)) \leq \lg(d) + \lg(g(k))$$

$$\lg(f(k)) \leq m \cdot \lg(g(k))$$

$$\text{So, } \lg(f(k)) = O(\lg(g(k)))$$

3.d. $f(k) = O(g(k))$ implies $2^{f(k)} = O(2^{g(k)})$

This statement is false, we can disprove as below:

Let $f(k) = 2n$ and $g(k) = n$

$$2n = O(n)$$

but $2^{2n} \neq O(2^n)$ because $4^n = O(n)$ but not $O(2^n)$

Theorem 4.1 (Master theorem)

Let $a \geq 1$ and $b > 1$ be constants, let $f(n)$ be a function, and let $T(n)$ be defined on the nonnegative integers by the recurrence

$$T(n) = aT(n/b) + f(n),$$

where we interpret n/b to mean either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Then $T(n)$ has the following asymptotic bounds:

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$.
3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n , then $T(n) = \Theta(f(n))$. ■

4.a. $T(n) = 2T\left(\frac{n}{2}\right) + n^4$

By comparing with the master theorem

$$a = 2, b = 2, f(n) = n^4$$

$$\text{So, } \log_b a = \log_2 2 = 1$$

Now, by considering case 3:

$$f(n) = \Omega(n^{\log_b a + \epsilon})$$

$$f(n) = n^4 = n^{1+3} \rightarrow \text{for } \epsilon = 3 \text{ and } \log_b a = 1$$

$$\text{and, } af(n/b) = 2f\left(\frac{n}{2}\right)$$

$$\text{if } f(n) = n^4 \Rightarrow f\left(\frac{n}{2}\right) = \frac{n^4}{8}$$

$$\text{So } af(n/b) = \frac{n^4}{8} \leq cn^4 \text{ for } \frac{1}{8} < c < 1 \text{ and } n \geq 1$$

$$\text{Then } T(n) = \theta(n^4)$$

4.b. $T(n) = T\left(\frac{7n}{10}\right) + n$

By comparing with the master theorem

$$a = 1, b = 10/7, f(n) = n$$

$$\text{So, } \log_b a = \log_{\frac{10}{7}} 1 = 0 = 1$$

Now, by considering case 3:

$$f(n) = \Omega(n^{\log_b a + \epsilon})$$

$$f(n) = n = n^{0+1} \text{ for } \epsilon = 1 \text{ and } \log_b a = 0$$

$$\text{and, } af(n/b) = 1 \cdot f\left(\frac{7n}{10}\right)$$

$$\text{So, } af(n/b) = f\left(\frac{7n}{10}\right) \leq cn \text{ for } \frac{7}{10} < c < 1 \text{ and } n \geq 1$$

Then, $T(n) = \theta(n)$

4.f. $T(n) = 2T\left(\frac{n}{4}\right) + \sqrt{n}$

By comparing with the master theorem

$a = 2, b = 4, f(n) = \sqrt{n}$

So, $\log_b a = \log_4 2 = 1/2$

Now, by considering case 2:

$f(n) = \theta(n^{\log_b a})$

$f(n) = \sqrt{n} \rightarrow \text{for } \log_b a = 1/2$

Then $T(n) = \theta(\sqrt{n} \lg n)$

4.g. $T(n) = T(n-2) + n^2$

we cannot apply master theorem

$T(n) = T(n-2) + n^2$ -eq.1

Let $n = (n-2)$

$T(n-2) = T((n-2)-2) + (n-2)^2 = T(n-4) + (n-2)^2$ -eq.2

Let $n = (n-4)$

$T(n-4) = T((n-4)-2) + (n-4)^2 = T(n-6) + (n-4)^2$ -eq.3

Let $n = (n-6)$

$T(n-6) = T((n-6)-2) + (n-6)^2 = T(n-8) + (n-6)^2$ -eq.4

By substituting eq.2 in eq.1 we get,

$T(n) = T(n-4) + ((n-2)^2) + n^2$ -eq.5

By substituting eq.3 in eq.5 we get,

$T(n) = T(n-6) + (n-4)^2 + ((n-2)^2) + n^2$

By this we get,

$T(n) = T(0) + 2^2 + 4^2 + \dots + (n-4)^2 + (n-2)^2 + n^2$

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

$$T(n) = T(0) + \sum_{i=0}^{n/2} n^2 - \sum_{i=0}^{n/2} 4ni + \sum_{i=0}^{n/2} 4i^2$$

$$T(n) = T(0) + \theta(n^2) + \theta(n) + \theta(1)$$

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

5. Given a, c are positive constants such that

$$T(n) = \sum_{j=2}^n (a + (j-1)c)$$

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

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

We know that

$$\sum_{i=2}^n (j-1) = \frac{n(n-1)}{2} = \frac{n^2-n}{2} = \frac{n^2}{2} - \frac{n}{2} \quad \text{- eq.1}$$

Substituting eq.1 in T(n) we get

$$T(n) = \frac{n^2}{2}c - \frac{n}{2}c + a \quad \text{-eq.2}$$

Eq.2 is in the form of $T(n) = An^2 + Bn + c$

So, we get $A = c/2$, $B = c/2$, $C = a$

From this we can say that A, B, C depend on a and c but not on n.

When evaluating Time complexity $T(n) = O(n^2)$ we ignored actual statement costs and abstract costs of all the constants in eq.2 because we are only interested in order of growth of the running time.

We therefore consider only the leading term of the above formula i.e., An^2 since the lower order terms are relatively insignificant for large values of n.

If $A = 0$ then the value of n^2 doesn't exist which gives a different runtime value which makes whole equation false.

From this we can say that only if $A > 0$ the value of n^2 exist.

6. Given, if $\text{Mindepth}(t) \geq n$, then $\text{CountNil}(t) \geq 2^n$

By induction proof we can prove,

if $\text{mindepth}(t) \geq n+1$ then $\text{countnil}(t) \geq 2^{n+1}$

For example: if $n=1$

$\text{Mindepth}(t) \geq 2$ and $\text{countnil} \geq 2^1$

If $n=2$

$\text{Mindepth}(t) \geq 2$ and $\text{countnil} \geq 2^4$

For $\text{mindepth}(t) \geq n+1$ means $\text{mindepth}(\text{left}(t) \geq (n+1))$ or
 $\text{mindepth}(\text{right}(t) \geq (n+1))$

If $\text{mindepth}(\text{left}(t)) \geq n+1$ then $\text{countnil}(\text{left}(t)) \geq n+1$

Also, $\text{mindepth}(\text{right}(t)) \geq n+1$ then $\text{countnil}(\text{right}(t)) \geq n+1$

From this we can say that even after growing incrementally

$\text{mindepth}(t) \geq n+1$ then $\text{countnil}(t) \geq 2^{n+1}$

so by this it is also true that $\text{Mindepth}(t) \geq n$, then $\text{CountNil}(t) \geq 2^n$