Lamess Kharfan
Student #: 10150607
Feb 8, 2014

CPSC 331- Assignment 1

1) Loop Invariant: $0 \leq j \leq n$
   $0 \leq maxS \leq S$
   S is the maximal value of the sum $\sum_{k=1}^{j} A[k]$ for all possible values of i and j

   Loop Variant: $f(n, j) = n - j$

2) Loop Invariant: $0 \leq k \leq j$
   $S = A[i] + A[i + 1] + . . . + A[j]$ for $i \leq k \leq j$

Proof by Induction on k:

Base Case (k = i):

Before the first execution of the loop, k = i, so the loop test passes, implying that $i \leq k \leq j$, S = 0 as it is assigned to 0 in the step prior to the loop execution so we may add terms to it, and the array, A, is unchanged. Thus, the loop invariant hold for the base case.

Inductive hypothesis: Assume that the loop body is executed at least $k \geq i$ times, and the loop invariant is satisfied at the beginning of the kth execution.

By inspecting the code, we see k is increased by 1 at the end of the ith execution, and:

- $i \leq k \leq j$
- S = A[i]
- Array is unchanged

If there is a k + 1st execution of the loop body, the loop test must pass after the end of the kth execution, ($k \leq j$), implying immediately after the k + 1st execution:

- $i \leq k \leq j + 1$
- S = A[i] + A[i + 1]
- Array is unchanged

Thus, the loop invariant holds.

Therefore, by mathematical induction, since the loop invariant holds for the k + 1st iteration, it holds for each iteration from then on.

Now, we apply the loop invariant to prove the post conditions hold.
Once we have reached the end of the loop condition, $i \leq k \leq j$ $k \leq j$, fails. The following properties will be satisfied:

- k is an integer such that $i \leq k \leq j + 1$
- S = A[i] + A[i + 1] + . . . + A[j]
- Array is unchanged
- k = j + 1

The first 3 properties come from the termination condition (the loop guard), the last comes from the incrementing of k.
The post conditions are satisfied because:
When k = j + 1, the post condition that the output is S= A[i] + A[i +1] + . . . + A[j] is satisfied since the loop cannot be entered if k > j, which it is since k = j + 1, so the loop guard tests as false and the loop is entered, so no more terms are added to S.

To prove the loop terminates, we use the loop invariant, f(j, k) = j – k.
This is the loop invariant because:

- It is integer valued since both j and k are integers
- f(j, k) = j – k is decreased because k is incremented after each iteration of the loop, and j remains constant.
- f(j, k) ≤ 0 when j ≥ k, and the for loop terminates when k = j + 1.

3) An upper-bound for a function $T_1(n)$ that describes the worst-case number of steps executed by the algorithm given when the input array has n elements is:

$$T_1(n) = C_1(n^3) + C_2C_3(n^2) + C_4(n) + C_5$$

$T_1(n)$ was derived by analyzing the algorithm, starting from the worst case of the inner most loop and working outwards, like so:

Inner-most loop:

```
while (k > j)
        S = S + A[k] → 2 steps, 1 assignment and 1 arithmetic
        k = k + 1 → 2 steps, 1 assignment and 1 arithmetic
```

The body of the loop executes j – i + 1 times, and the loop guard is checked j – i + 2 times, and because the body of the loop consists of 4 steps, we get the equation:

4(j – i + 1) + (j – i + 2) = 5(j – i) + 6

We may replace j – i with n because the value of j – i is less than that of n, and since we are looking for an upper-bound it is appropriate to replace smaller variables with larger ones, such as n. So we get 5n + 6.
We then can generalize constants as these are not of extreme importance when generating an upper-bound, so we finally get, $C_1n + C_2$, where c1 and c2 are constants.

We repeat this process with the second inner-most loop:

```
while (i < j)
        S = 0 → 1 step, 1 assignment
        k = i → 1 step, 1 assignment
        ($C_1n + C_2$)
        if (S > maxS) → 1 step, 1 comparison
                maxS = S → 1 step, 1 assignment
        i = i + 1 → 2 steps, 1 arithmetic, 1 assignment
```

Lamess Kharfan
Student #: 10150607
Feb 8, 2014

If we follow the same process as above, where we form an equation with constants and generalize them into variable names, then add in the worst case of the inner loop, and replace the number of iterations with n, since n is larger than the number of iterations and it is appropriate to switch n in when calculating an upper bound. We get the equation:

$$n((C_1 n + C_2) + C_3) + C_4$$
$$= C_1(n^2) + C_2C_3(n) + C_4$$

And once again, we move to the outermost loop and repeat the above process by first, counting the steps, the recognizing that the loop goes through exactly n iterations, then adding the equation from above, which is the worst case of the inner-loops combined, and replace the constants with variable names, and end up with the equation:

$$n(C_1(n^2) + C_2C_3(n) + C_4) + C_5$$
$$= C_1(n^3) + C_2C_3(n^2) + C_4(n) + C_5$$

4) Proof that $T_1(n) \in O(n^d)$
Let $f(n) = C_1(n^3) + C_2C_3(n^2) + C_4(n) + C_5$ and $g(n) = n^3$.
Then,
$f(n) = C_1(n^3) + C_2C_3(n^2) + C_4(n) + C_5 \leq C_1(n^3) + C_2C_3(n^3) + C_4(n^3) + C_5(n^3) = d(n^3),\ d = C_1 + C_2C_3 + C_4 + C_5$ for $n^3 \geq 3, n = \sqrt[3]{3}\ (N_0)$
Thus, $f(n) \leq g(n)\ \forall n \in N_0$ when $c = d$ and $N_0 = \sqrt[3]{3}$.
So, by definition, $f \in O(g)$

5)

$$T_2(n) \begin{cases} 3 \text{ units} & n = 0 \\ 5 \text{ units} & n = 1 \\ 14 + T_2(\lceil n/2 \rceil + 1) + T_2(\lceil n/2 \rceil - n + 1) + c_1\theta(n) + c_2\theta(n), & n \geq 2 \end{cases}$$

Recursive Call S$_L$: Suppose $j \in \left\{0, . . . , \left(\frac{n}{2}\right)\right\}$, then the array is of length n/2 – 0 +1 = n/2 + 1. It appropriate to take the ceiling of the value of n/2 since we are calculating the worst case number of steps when the array has n elements, and n/2 < , and a "step" is a whole number, not a fraction. So we end up with $T_2\left(\left\lceil\frac{n}{2}\right\rceil + 1\right)$

Recursive Call S$_R$: Suppose $j \in \left\{(\frac{n}{2} + 1, . . . , n - 1\right\}$, then the array is of length n-1 – (n/2) + 1 = n- n/2 -1, and again, it is appropriate to take the ceiling of n/2 for the same reasons indicated for recursive call S$_R$. The result is the function $T_2\left(\left\lceil\frac{n}{2}\right\rceil - n + 1\right)$

Lamess Kharfan
Student #: 10150607
Feb 8, 2014

S_1 and S_2: As already mentioned above, the length of the arrays are n/2 +1 and n – n/2 -1, and since these are linear functions of n, they may be replaced with Θ(n), so we get c1Θ(n) + c2Θ(n).

6) Relation between T1(n) and T2(n), T1(n) and T3(n), T2(n) and T3(n)

Relation between T1(n) and T2(n): $n^3 \in w(nlogn)$ since n^3 grows strictly faster than nlogn

Relation between T1(n) and T2(n): $n^3 \in w(n)$ since n^3 grows strictly faster than n

Relation between T2(n) and T3(n): $nlogn \in o(n)$ because nlogn grows strictly slower than n.

7) Black box test for the maximum subsequence problem:

Input	Expected Output	Purpose
Null Array: A = null	IllegalArgumentException	Invalid Input
Empty Array: A=[ ]	Max subsequence sum = 0	Typical Case
1 element Array: A = [x]	Max subsequence sum = x	Boundary Case
Max sequence sum is sum of last 2 elements in array: A= [….., x, y]	Max subsequence sum = x +y	Boundary Case
Max Subsequence sum is the sum of the first two elements in the array: A = [w, z, …..]	Max subsequence sum = w + z	Boundary Case
All negative integer entries	Max subsequence sum = 0	Typical case
Non-integer array	NoSuchElementException	Invalid Input
Integer array	$\sum_{k=1}^{j} A[k]$ for all possible values of i and j	Typical Case

Lamess Kharfan
Student #: 10150607
Feb 8, 2014

8) A set of white-box tests for the function maxSubSum3:
   Test for when:
   - A = null: will throw the IllegalArugmentException and print an error message
   - A = an integer array: no exceptions thrown, proceed to next lines.
   - A.length > MAX_ARRAY_SIZE: will throw the IllegalArgumentException and print an error message.
   - A.length <= MAX_ARRAY_SIZE: no exceptions thrown, proceed to next lines.
   - j >= A.length - 1: loop guard is false, skip over for loop, assert maxSum, return maxSum.
   - j < A.length -1: loop guard is true, move into the loop body
   - thisSum <= maxSum: if condition is false, skip over, proceed to else if.
     - thisSum >= 0: else if statement is false, skip over else-if body. Assert maxSum. Return maxSum.
     - thisSum < 0: else if statement is true, proceed into the body of the else-if. Assert maxSum, return maxSum.
   - thisSum > maxSum: if statement if true, move into body of if statement, skip over else if statement, assert maxSum, return maxSum.

9) See Junit Test harness named MSSTester1.java, MSSTester2.java, MSSTester3.java, MSSTester4.java for all three implementations of the black tests and the implementation of the white box test.
10) See MaxSubsequenceSum.java