Elijah Chou

CS 326

Dr. Michelangelo Grigni

2/5/2022

HW 2

1. (a): For n<10, the property holds since n < n +1. Even so, the case where T(10) ≤ T(11) must be checked.

This shows that T(11) is indeed greater than or equal to T(10). T(11) ≤ T(12) will also be proven here just to be safe.

Now that T(11) ≤ T(12) is shown to be true, the base case is now concluded.

Now we assume that for all *i,* where 1 ≤ *i* ≤ *k*, for some *k* ≥ 11, T(*i* – 1) ≤ T(*i*) holds. To prove that T(*k*) ≤ T(*k*+1), I will prove two cases: one where *k* + 1 is odd and another where *k* + 1 is even.

Case 1: when *k* + 1 is odd

Case 2: when *k* + 1 is even

We can show that (*k*+1)/2 + 5 ≤ *k* 🡪 *k* + 11 ≤ 2*k* 🡪 *k* ≥ 11, and we know that these equal statements are true since one of the assumptions made earlier is that *k* ≥ 11. By the Strong Inductive argument, we have the following:

From this information, we can deduce the following:

Thus, we have proven that T(*k*) + 1 ≥ T(*k*).

(b): Starting with base case 11 = 20 + 11, we get:

This is true for some *c* and *d*. Assuming that this also works for *k*, we get that T(2k + 10) ≤ 2k(*k* + *c*) – *d*. Then, we need to prove this for 2k+1 + 10:

Then:

With the constraint of *d*, we can choose an arbitrary value of *d* and a corresponding value of *c* to get a final equation. Choosing *d* = 10 and *c* = 41, we finally get that T(2k + 10) ≤ 2k (*k* + 41) – 10.

(c): Now that we know that T(2k + 10) ≤ 2k (*k* + 41) – 10, for values *n* in between 2k + 10 and 2k+1 + 10, we know that T(*n*) ≤ T(2k+1 + 10) ≤ 2k+1 (*k* + 41) – 10 based on parts (a) and (b). Since *n* = 2k + 10 🡪 *k* = lg *n* – 1g 10, we get that:

Since we ignore constant terms and lesser terms such as 41*n* and 410, we get that T(n) ∈ O(n lg n).

1. ith(A, B, aStart, bStart, aEnd, bEnd, i)

if (aStart == aEnd) return B[k]

if (bStart == bEnd) return A[k]

aMedian = (aEnd – aStart) / 2

bMedian = (bEnd – bStart) / 2

if (k > aMedian + bMedian)

if (A[aMedian] > B[bMedian])

return ith(A, B, aStart, bStart + bMedian + 1, aEnd, bEnd, k – bMedian – 1)

else

return ith(A, B, aStart + aMedian + 1, bStart, aEnd, bEnd, k – aMedian – 1)

else

if (arr1[aMedian] > arr2[bMedian])

return ith(A, B, aStart, bStart, aStart + aMedian, bEnd, k)

else

return ith(A, B, aStart, bStart, aEnd, bStart + bMedian, k)

This algorithm functions by repeatedly removing half of one of the arrays each time the ith function is called. We know that array A can only be halved lg(*m*) times and array B can only be halved lg(*n*) times. There may be cases when one array does not get halved, but the worst case is when both of the arrays are halved the maximum number of times, which would add up to be lg(*m*) + lg(*n*). Since lg(m) + lg(n) ∈ O(lg(m+n)), this algorithm also has O(lg(m+n)).

1. PRINT-LCS(c, X, Y, i, j)

if i = 0 or j = 0

return

if X[i] == Y[j]

PRINT-LCS(c, X, Y, i – 1, j – 1)

print X[i]

elseif c[i – 1, j] >= c[i, j – 1]

PRINT-LCS(c, X, Y, i – 1, j)

Else

PRINT-LCS(c, X, Y, i, j – 1)

1. A picture containing text, whiteboard

   Description automatically generated  
   A picture containing shape

   Description automatically generated
2. Program included in Canvas submission. The output is as follows:  
   n: 9177855

S(n): 101745647

M(n): 203491305