Algorithms HW2 *Summer 2019* Tae Myles

1 A)

// Dividing each side by n

// Plugging in larger and larger values for n makes it smaller, so,

f(n) = Θ(g(n))

B) c

// By identity

f(n) = Θ(g(n))

C)

// n grows faster, so switching signs.

f(n) = Ω(g(n))

D)

// Polynomial grows faster than logarithm

f(n) = Ω(g(n))

E) )

// Exponential functions grow faster

f(n) = O(g(n))

2 A) Mystery Algorithm finds the smallest value of elements in the array.

B) Algorithm Mystery(A: Array [i..j] of integer)

i & j are array starting and ending indexes

begin

if i=j then return A[i] **7**

else

k=i+floor((j-i)/2) **8**

temp1= Mystery(A[i..k]) **5**

temp2= Mystery(A[(k+1)..j] **6**

if temp1<temp2 **3**

then return temp1 **2**

else return temp2 **2**

end

// Numbers that are bold are the cost

T(n) = { 7 , n = 1}

{ 2T(n/2) + 29 otherwise}

T(n) = { 7 , n = 1}

{ 2T(n/2) + 33 otherwise}

C)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Level | Level Number | Total # of recursive executions at this level | Input size to each recursive execution | Work done by each recursive execution, excluding the recursive calls | Total work done by the algorithm at this level |
| Root | 0 | 1 | n | cn | cn |
| One level below the root | 1 | 2 | n/2 | C(n/2) | Cn |
| Two levels below root | 2 | 4 | n/4 | C(n/4) | Cn |
| The level just above the base case level | logn-1 | n-1 |  |  |  |
| Base case level | logn | n | 1 | c | cn |

D)

3.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Level | Level Number | Total # of recursive executions at this level | Input size to each recursive execution | Work done by each recursive execution, excluding the recursive calls | Total work at this level |
| Root | 0 | 1 | n | cn | Cn |
| 1 level below | 1 | 7 | n/8 | C(n/8) | Cn(7/8) |
| 2 levels below | 2 | 49 | n/64 | C(n/64) |  |
| The level just above the base case level | Log8n - 1 | 7log8n-1 | 8 | 8c | 8n-2 |
| Base case level | Log8n | 7log8n | 87) | 1 | cn87) |

T(n) = 87)

// From the x to the I summation formula

T(n) = 87)

T(n) = 87)

4. Statement of what you have to prove:

T(n) = nlogn + 5n

Base case proof:

Let n = 1, then T(1) = (1)log(1) + 5(1) = 5, which holds.

Inductive Hypotheses:

Prove T(n) c(nlogn) + cn for c < n where c is a constant

Inductive Step:

T(n) = 3T(n/3) + 5

= c(nlogn(n/3)) + 5

= c(nlogn) – c(nlog3) + 5

= c(nlog3n) – cn + 5

Value of c:

c >

5. If f(n) = O(s(n)) then there exists a constant c and n0 such that for all n n0, 0 f(n) cg(n).

If g(n) = O(r(n)) then there exists a constant c and n0 such that for all n n0, 0 f(n) cg(n).

So, by definition, we can try to plug in corresponding variable for the given imply,

f(n) = , s(n) = , g(n) = n, r(n) = n

Given, f(n) – g(n) = O(s(n) – r(n)) ⬄ = O(n - n)

As shown above, we can easily see that it will lead to contradiction, because trying any value for n n0 will lead to O(s(n) – r(n)) not being a big-o of f(n) - g(n).

6. (1) T(n)

/ / \ \

/ / \ \

T(n/2) T(n/4) T(n/8) T(n/8)

/ / \ \ / / \ \ / / \ \ / / \ \

T(n/4) T(n/8) T(n/16) T(n/16) T(n/8) T(n/16) T(n/32) T(n/32) T(n/16) T(n/32) T(n/64) T(n/64) T(n/32) T(n/64) T(n/128) T(n/128)

A) Level 0: 1

Level 1: 4

Level 2: 8

B) Level 0: n

Level 1: n/2, n/4, n/8, n/8

Level 2: n/4, n/8, n/16, n/16

C) Level 0: n

Level 1: cn

Level 2: cn

D) Level 0: cnn

Level 1: cn(n/2 + n/4 + n/8 + n/8)

Level 2: cn(n/4 + n/8 + n/16 + n/16 + n/8 + n/16 + n/32 + n/32 + n/16 + n/32 + n/64 + n/64 + n/16 + n/32 + n/64 + n/64)

(2) Shown above

(3) Log2n

(4) Log8n

(5) The big-o is at most O(n), the reason for this guess is because sum of recursive calls are

7. Statement of what you have to prove:

T(n) = O(n) for T(n) = T(n\2) + T(n\4) + T(n/8) + T(n/8) + n

T(1) = c

Base case proof:

For c > 0, set n = 1 and c = 1.

Then, T(1) = (1)(1) = 1, it holds

Inductive Hypotheses:

Prove T(n) cT(n\2) + cT(n\4) + cT(n/8) + cT(n/8) + cn for c < n where c is a constant

Inductive Step:

T(n) = T(n\2) + T(n\4) + T(n/8) + T(n/8) + n

= c(n/2) + c(n/4) + c(n/8) + c(n/8) + n

= cn) + n

= n(c) + 1)

8. A) T(n) = 2T(99n/100) + 100n

a = 2, b = 100/99 = 1.01

nlogba = nlog(1.01) = n68.9

100n = 0( where > 0

T(n) =

B) T(n) = 16T(n/2) + n3lgn

a = 16, b = 2

216 = n4

f(n) = n3logn

n3logn = O(216 – ε) where > 0

T(n) = (n4)

C) T(n) = 16T(n/4) + n2

a = 16, b = 4

nlogba = n2

f(n) = n2

9. (1) T(n) = 2T(n-1) + 1

T(n-1) = 4T(n-2) + 2 + 1

T(n-2) = 8T(n-3) + 4 + 2 + 1

T(n-3) = 16T(n-4) + 8 + 4 + 2 + 1

(2) T(n) = 2iT(n – i) + 2i-1 + 2i-2 + …

Hence, T(n) = 2(n+1) – 1

(3) // We know from backward substitution that LHS is T(n) = 2(n+1) – 1, now to find RHS

T(n) = 2T(n-1) + 1 , t(0) = 1

T(1) = 2T(1-1) + 1 = 3

T(2) = 2T(2-1) + 1 = 7

T(3) = 2T(3-1) + 1 = 15

Hence, RHS is T(n) = 2(n+1) – 1 and from above, LHS, which is T(n) = 2(n+1) – 1. It equals!

(4) Big-o time complexity is O(2^n)

10. T(n) = T(n-1) + n/2, T(1) = 1

T(n-1) = T(n-2) + ((n-1)/2) + n/2

T(n-2) = T(n-3) + ((n-2)/2) + ((n-1)/2) + n/2

T(n-3) = T(n-4) + ((n-3)/2) + ((n-2)/2)) + ((n-1)/2)) + n/2

T(n-i) = T(n-i) + ((n – i + 1)/2) + ((n – i + 2)/2)

Hence,

T(n) = T(1) + ( ) / 2

= 1 + (2 + 3 + … + n)/2

= 1 + (1 + 2 + .. + n – 1)/2

// From the summation formula i

= 1 +

= 1 +

= 1 +

=

=

11. 2n, T(2) = 4

T(4) = 2T(24 = 2T(2) + 16 = 8 + 16 = 24

T(8) = 2T(8/2) + 16log28 = 2T(4) + 48

T(n) = + 6log2n = O(nlog2n)

12. It wouldn’t make sense for running algorithm to be at least O(n2). It should be at most, since big-o sets the upper bound.