Marco Zuniga

COMP 3270

HW 2

June 12, 2018

1.
   a. $f(n) = \Omega(g(n))$
   b. $f(n) = O(g(n))$
   c. $f(n) = \Omega(g(n))$
   d. $f(n) = \Omega(g(n))$
   e. $f(n) = O(g(n))$

2.

```
Algorithm Mystery(A: Array [i..j] of integer)
i & j are array starting and ending indexes
 begin
   if i=j then return A[i]
   else
     k=i+floor((j-i)/2)
     temp1= Mystery(A[i..k])
     temp2= Mystery(A[(k+1)..j]
     if temp1<temp2 then return temp1 else return temp2
  end
```

(a) The smallest number in the input array.

(b) **$T(n) = 7$ for $n \leq 1$**

Get i
Get j
Check if equal
Get i
Find memory location of i
Get value at memory location
Return

$T(n) = 2T(n/2) + 18$

Get i
Get j
Check if equal
Get j
Get i
Subtract
Divide by 2
Run floor
Get i

Do addition
Assign to k
Assign to temp1
Assign to temp2
Get temp1
Get temp2
Make comparison
Get either temp1 or temp2
Return

(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	c	c
One level below root	1	2	n/2	c	2c
Two levels below	2	4	n/4	c	4c
The level just above the base case level	$\log_2(n - 1)$	$2^{\log_2 n - 1}$	$n/(2^{\log_2 n - 1})$	c	$2^{\log_2 n - 1} * C$
Base Case Level	$\log_2(n)$	$2^{\log_2 n}$	1	c	$2^{\log_2 n} * C$

(d) $T(n) = \sum_{i=0}^{\log_2 n-1} 2^i c + n^{\log_2 2} c$

$< \sum_{i=0}^{\infty} 2^i c + nc$

$= c(1/(1-2)) + nc$

$T(n) = O(n)$

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	$c\frac{n}{8}$	(7/8)cn
2 levels below	2	49	n/64	$c\frac{n}{64}$	(49/64)cn
The level just above the base case level	$\log_{8(}n) - 1$	$7^{\log_8 n - 1}$	$n/(8^{\log_8 n - 1})$	$cn/(8^{\log_8 n - 1})$	$7^{\log_8 n - 1}*cn/(8^{\log_8 n - 1})$
Base case level	$\log_{8(}n)$	$7^{\log_8 n}$	1	$cn/(8^{\log_8 n})$	$7^{\log_8 n} * cn/(8^{\log_8 n})$

$T(n) = [cn + (7/8)cn + (49/64)cn + \ldots. + 7^{\log_8 n - 1}*cn/(8^{\log_8 n - 1})] + 7^{\log_8 n} * cn/(8^{\log_8 n})$

$= [cn + (7/8)cn + (49/64)cn + \ldots. + 7^{\log_8 n - 1}*cn/(8^{\log_8 n - 1})] + n^{\log_8 7} * cn/(n^{\log_8 8})$

$= [cn + (7/8)cn + (49/64)cn + \ldots. + 7^{\log_8 n - 1}*cn/(8^{\log_8 n - 1})] + n^{\log_8 7} * cn/(n)$

$= [cn + (7/8)cn + (49/64)cn + \ldots. + 7^{\log_8 n - 1}*cn/(8^{\log_8 n - 1})] + n^{\log_8 7} * c$

$= [cn + (7/8)cn + (49/64)cn + \ldots. + 7^{\log_8 n - 1}*cn/(n/8)] + n^{\log_8 7} * c$

4. By using the substitution method, the student will try to prove the correct time complexity of the following recurrence relations: $T(n) = 3T(n/3) + 5$; $T(1) = 5$.
Statement of what you have to prove:
By using the substitution method the student will guess $T(n) = O(n)$ as well as use proof by induction to show $T(n) \leq cn$
Base Case Proof:
$5 = T(1) \leq c*1$
$= 5 \leq c*1$ if $c \geq 5$
Inductive Hypothesis:
Now assume $T(n/3) \leq c[n/3]$

Inductive Step:
We must show $T(n) \le cn$
$T(n) \le 3c[n/3] + 5 \le cn + 5$
$\le cn$ ***Proof fails**

2^{nd} Try

Statement of what you have to prove:
By using the substitution method the student will guess $T(n) = O(n\log_3 n)$ as well as use proof by induction to show $T(n) \le cn\log n$

Base Case Proof:
$5 = T(1) \le c1\log(1)$
$= 5 \le c*0$
However, $T(3) = 20 \le c3\log(3)$ if $c > 6.66$

Inductive Hypothesis:
Now assume $T(n/3) \le c[n/3]\log[n/3]$

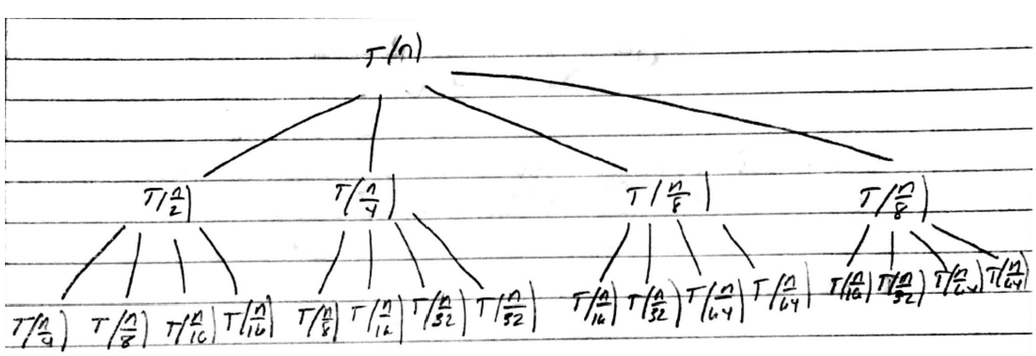
Inductive Step:
We have to show $T(n) \le cn\log n$
$T(n) \le 3(c[n/3]\log[n/3] + 5 \le cn\log(n/3) + 5$
$= cn\log n - cn\log 3 + 5$
$= cn\log n - cn + 5$
$\le cn\log n$ if $c \ge 5$ and $n \ge 1$.

Value of c:
$c \ge 5$

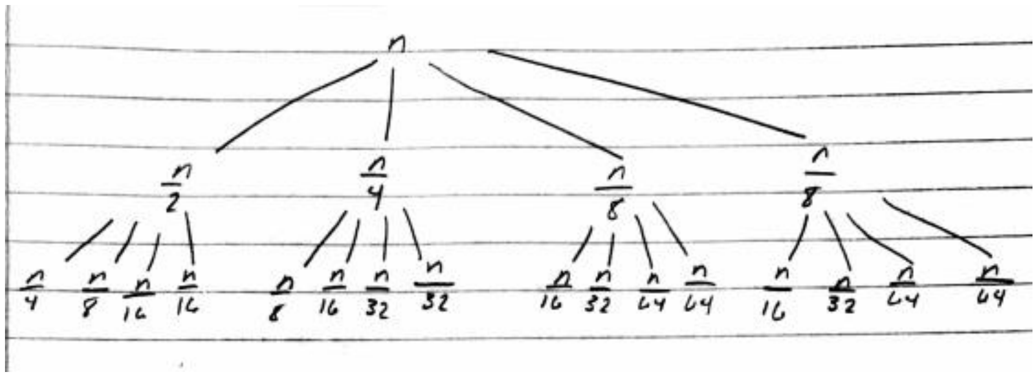
5. The student was required to find a counterexample to the following claim.
$f(n)=O(s(n))$ and $g(n)=O(r(n))$ imply $f(n) - g(n) = O(s(n) - r(n))$
Assume $s(n) = n^2$ and $r(n) = n$.
$f(n) - g(n)$ implies that $O(s(n)) - O(r(n)) = O(s(n) - r(n))$
Plugging in the values we get $O(n^2) - O(n) = O(n^2 - n)$
$= O(n^2)$
$O(n^2) - O(n) \ne O(n^2)$
Therefore, the statement is proved to be incorrect.

6. $T(n) = T(n/2) + T(n/4) + T(n/8) + T(n/8) + n; T(1) = c$

(a)



Recursive Tree with Recursive Execution At Each Level



Recursive Tree with Work At Each Recursive Execution - Levels 0-2

(b) Input size to each recursive execution
Level 0: n
Level 1: n/2, n/4, n/8, n/8
Level 2: 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

(c) Work done by each recursive execution
Shown in second figure.

(d) Total work down at each level
Level 0: n
Level 1: [n/2 + n/4 + n/8 + n/8] = n
Level 2: [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] = n

(2) Shown above in Figure

(3) Depth of tree at its shallowest part
$\log_8(n)$; The input size is recursively being divided by 8 so it will reach the base case the fastest.

(4) Depth of tree at its deepest part
$\log_2(n)$; The input size is recursively being divided by 2 so it will take the longest to reach the base case.

(5) O(n)

7. By using the substitution method, the student was required to prove the guess from the previous question is correct.
<u>Statement of what you have to prove</u>:
By using the substitution method, the student will guess T(n) = O(n) as well as use proof by induction to show $T(n) \leq dn$

Base Case Proof:
T(1) = c ≤ dn if d ≥ c
Inductive Hypothesis:
Now assume T(n/2) ≤ c(n/2)
T(n/4) ≤ c(n/4)
T(n/8) ≤ c(n/8)
T(n/8) ≤ c(n/8)
Inductive Step:
We must show T(n) ≤ dn
T(n) ≤ c(n/2) + c(n/4) + c(n/8) + c(n/8) + c
= c + cn
≤ dn if d ≥ (c/n) + c

8.

(a) T(n) = 2T(99n/100) + 100n
a = 2, b = 100/99, f(n) = 100n
Case 1
- **T(n) = theta(n^$^{\log_{100/99} 2}$) = theta($n^{68.9676}$)**

(b) T(n) = 16T(n/2) + n^3logn
a = 16, b =2, f(n) = n^3logn
Case 1
- **T(n) = theta(n^4)**

(c) T(n) = 16T(n/4) + n^2
a = 16, b = 4, f(n) = n^2
Case 2
- **T(n) = theta(n^($\log_b$a) * logn) = theta(n^2 * logn)**

9. By using the Backward and Forward substitution, the student was required to solve the recurrence relations; T(n) = 2T(n – 1) + 1; T(0) = 1 using $\sum_{k=0}^{n} x^k = \frac{x^{n+1}-1}{x-1}$.

Backward Substitution

(1) Three Expansions
T(n – 1) = 2T(n – 2) + 1
T(n) = 4T(n – 2) + 2 + 1
T(n – 2) = 2T(n – 3) + 1
T(n) = 8T(n – 3) + 4 + 2 + 1
T(n – 3) = 2T(n – 4) + 1
T(n) = 16T(n – 4) + 8 + 4 + 2 + 1
By looking at the three expansions, one can see that the Recursive relations will just compute the summation of 2^i from 0 to n. Once it reaches the base case T(0), which is 1, it will be multiplied by the last number needed to be added. Using the equation provided, 2 = x and i = k.

$$\sum_{k=0}^{n} 2^k = \frac{2^{n+1}-1}{2-1} = 2^{n+1} - 1 = \text{LHS}$$

(2) LHS = T(n) = $2^{n+1} - 1$
RHS = 2T(n – 1) + 1 = $2(2^{n-1+1} - 1)$ - 1 = $2^{n+1} - 2 - 1 = 2^{n+1} - 1$

(3) LHS = RHS

(4) T(n) = $O(2^n)$

Forward Substitution

(1) Three Expansions
T(1) = 2T(0) + 1 = 3
T(2) = 2T(1) + 1 = 7
T(3) = 2T(2) + 1 = 15
One can see that that T(n) = $2^{n+1} - 1$, which is what was found by using backward substitution.

(2) LHS = T(n) = $2^{n+1} - 1$
RHS = 2T(n – 1) + 1 = $2(2^{n-1+1} - 1) + 1 = 2^{n+1} - 1$

(3) LHS = RHS

(4) T(n) = $O(2^n)$

10. Here the student was required to solve the following recurrence relations;
T(n) = T(n – 1) + n/2; T(1) = 1
The student used forward substitution to see a pattern.
T(1) = 1
T(2) = 2
T(3) = 7/2
T(4) = 11/2
T(5) = 16/2
T(6) = 22/2
The student couldn’t see a precise pattern, so he put all values with the same denominator
T(1) = 4/4
T(2) = 8/4
T(3) = 14/4
T(4) = 22/4
T(5) = 32/4
T(6) = 44/4
The student began to bring the n values to different exponents. First by squaring all n values, the result will come out close to the correct output. After trying to raise n to the 3rd and 4th power, one will find that the number will eventually be far from the correct solution. The addition of the n value squared, the n value, and 2, the value will always come out to the numerator results. Since they are all over 4, the final equation found was
T(n) = $(n^2 + n + 2)/4$
T(n) = $O(n^2)$

11. The student was required to prove that T(n), which is defined by the recurrence relation
$T(n) = 2T([n/2]) + 2n\log_2 n$; $T(2) = 4$
satisfies $T(n) = O(n\log^2 n)$
To prove this, the student used the substitution method to prove $T(n) = O(n\log^2 n)$.
Statement you have to prove:
By using the substitution method, the student will guess $T(n) = O(n\log^2 n)$ as well as use induction to prove $T(n) \leq cn\log^2 n$
Base Case:
$T(2) = 4 \leq c2\log^2 2$
$4 \leq c2*1\text{^}2$ if $c \geq 2$
Inductive Hypothesis:
Now assume $T(n/2) \leq c(n/2)\log^2(n/2)$
Inductive Step:
We must show $T(n) \leq cn\log^2 n$
$T(n) \leq 2(c(n/2)\log^2(n/2) + 2n\log_2 n$
$\leq cn\log^2(n/2) + 2n\log_2 n$
Here the student picked an example to help find the constant. By letting $n = 8$,
You'll get $c4 + 6 \leq c9$ if $c \geq 2$
Thus, $T(n) \leq cn\log^2 n$
$T(n) = O(n\log^2 n)$

12. *Big O* is used to represent worst time complexity/the upper bound. We need to state that the running time of algorithm is at least $\Omega(n^2)$ which is the lower bound.