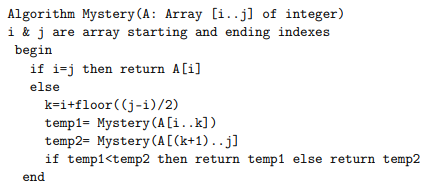
Marco Zuniga

COMP 3270

HW 2

June 12, 2018

1. *f(n)* = *theta(g(n))*
2. *f(n)* = *O(g(n))*
3. *f(n)* = *Ω(g(n))*
4. *f(n)* = *Ω(g(n))*
5. *f(n*) = *O(g(n))*



1. The smallest number in the input array.
2. **T(n) = 7 for n ≤ 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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 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 | log2(n - 1) | 2log2n - 1 | n/(2log2n -1 ) | c | 2log2n – 1 \* C |
| Base Case Level | log2(n) | 2log2n | 1 | c | 2log2n \* C |

1. T(n) =

<

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

T(n) = *O(n)*



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 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 | (7/8)cn |
| 2 levels below | 2 | 49 | n/64 | c | (49/64)cn |
| The level just above the base case level | log8(n) - 1 | 7log8n - 1 | n/(8log8n - 1) | cn/(8log8n - 1) | 7log8n - 1\*cn/(8log8n - 1) |
| Base case level | log8(n) | 7log8n | 1 | cn/(8log8n) | 7log8n \* cn/(8log8n) |

T(n) = [cn + (7/8)cn + (49/64)cn + …. + 7log8n - 1\*cn/(8log8n - 1)] + 7log8n \* cn/(8log8n)

= [cn + (7/8)cn + (49/64)cn + …. + 7log8n - 1\*cn/(8log8n - 1)] + nlog87 \* cn/(nlog88)

= [cn + (7/8)cn + (49/64)cn + …. + 7log8n - 1\*cn/(8log8n - 1)] + nlog87 \* cn/(n)

= [cn + (7/8)cn + (49/64)cn + …. + 7log8n - 1\*cn/(8log8n - 1)] + nlog87 \* c

= [cn + (7/8)cn + (49/64)cn + …. + 7log8n - 1\*cn/(n/8)] + nlog87 \* c

1. 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) ≤ cn

Base Case Proof:

5 = T(1) ≤ c\*1

= 5 ≤ c\*1 if c ≥ 5

Inductive Hypothesis:

Now assume T(n/3) ≤ c[n/3]

Inductive Step:

We must show T(n) ≤ cn

T(n) ≤ 3c[n/3] + 5 ≤ cn + 5

≤ cn **\*Proof fails**

**2nd Try**

Statement of what you have to prove:

By using the substitution method the student will guess T(n) = O(nlog3n) as well as use proof by induction to show T(n) ≤ cnlogn

Base Case Proof:

5 = T(1) ≤ c1log(1)

= 5 ≤ c\*0

However, T(3) = 20 ≤ c3log(3) if c > 6.66

Inductive Hypothesis:

Now assume T(n/3) ≤ c[n/3]log[n/3]

Inductive Step:

We have to show T(n) ≤ cnlogn

T(n) ≤ 3(c[n/3]log[n/3] + 5 ≤ cnlog(n/3) + 5

= cnlogn – cnlog3 + 5

= cnlogn – cn + 5

≤ cnlogn if c ≥ 5 and n ≥ 1.

Value of c:

c ≥ 5

1. 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) = n2 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(n2) – O(n) = O(n2 – n)

= O(n2)

O(n2) – O(n) ≠ O(n2)

Therefore, the statement is proved to be incorrect.

Wrong ^. Professor Solution Below

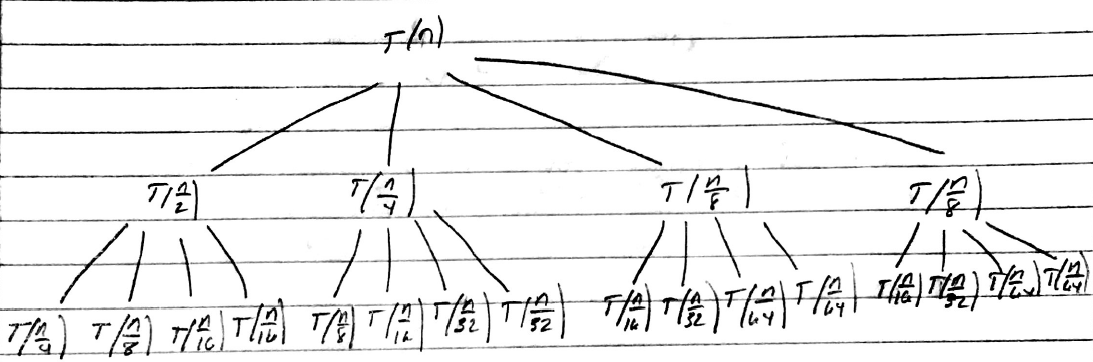
Let f(n) = 3n, s(n) = n, g(n) = 2n, and r(n) = n.

f(n) – g(n) = 3n – n = 2n

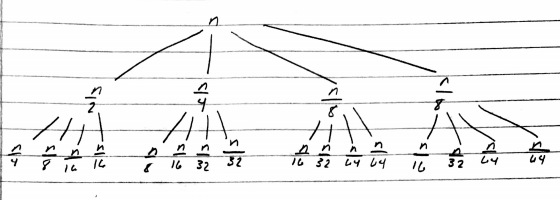
s(n) – r(n) = 0

2n ≠ 0

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



**Recursive Tree with Recursive Execution At Each Level**



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

Feedback:

It looks more like an acute angle triangle with the longer side showing the deepest part and shorter side showing the shallowest part.  
The tree in your figure shows how the input breaks down at every level but it doesnt clearly show that the tree has a varying depth.

1. 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

1. Work done by each recursive execution

Shown in second figure.

1. 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

1. Shown above in Figure
2. Depth of tree at its shallowest part

log8(n); The input size is recursively being divided by 8 so it will reach the base case the fastest.

1. Depth of tree at its deepest part

log2(n); The input size is recursively being divided by 2 so it will take the longest to reach the base case.

1. O(nlog(n)) because you’ll have some c times n when you add up all the work. So not just n. Must be nlog(n).
2. By using the substitution method, the student was required to prove the guess from the previous question is correct.

Statement of what you have to prove:

By using the substitution method, the student will guess T(n) = O(nlogn) as well as use proof by induction to show T(n) ≤ dnlogn

Base Case Proof:

T(1) = c ≤ dnlogn if d ≥ 1 and n ≥ 1

Inductive Hypothesis:

Now assume T(n/2) ≤ c(n/2log(n/2)), T(n/4) ≤ c(n/4log(n/4)), T(n/8) ≤ c(n/8log(n/8)), and

T(n/8) ≤ c(n/8log(n/8))

Inductive Step:

We must show T(n) ≤ dnlogn

T(n) ≤ c(n/2)log(n/2) + c(n/4)log(n/4) + c(n/8)log(n/8) + c(n/8)log(n/8) + n

= c(n/2)[log(n) - log(2)] + c(n/4)[log(n) – log(4)] + c(n/8)[log(n) – log(8)] + c(n/8)[log(n) – log(8)] + n

cnlogn – 2c(n/4) - c(n/4)\*2 - c(n/4)\*3 + n

= cnlogn + c(n/4)[-2 – 2 – 3]

= cnlogn – 7c(n/4) + n

= cnlogn – n(c7/4 – 1)

≤ dnlogn if d ≥ 1

1. T(n) = 2T(99n/100) + 100n

a = 2, b = 100/99, f(n) = 100n

**Case 1**

* **T(n) = theta(n^log100/992) = theta(n68.9676)**

1. T(n) = 16T(n/2) + n3logn

a = 16, b =2, f(n) = n3logn

**Case 1**

* **T(n) = theta(n4)**

1. T(n) = 16T(n/4) + n2

a = 16, b = 4, f(n) = n2

**Case 2**

* **T(n) = theta(n^(logba) \* logn) = theta(n^2 \* logn)**

1. 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 .

**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 2i 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.

= 2 n + 1 – 1 = LHS

1. LHS = T(n) =2n + 1 – 1

RHS = 2T(n – 1) + 1 = 2(2n – 1 + 1 – 1) - 1 = 2n + 1 – 2 – 1 = 2n + 1 – 1

1. LHS = RHS
2. T(n) = O(2n)

**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) = 2n + 1 – 1, which is what was found by using backward substitution.

1. LHS = T(n) = 2n + 1 – 1

RHS = 2T(n – 1) + 1 = 2(2n - 1 + 1 – 1) + 1 = 2n + 1 – 1

1. LHS = RHS
2. T(n) = O(2n)
3. 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) = (n2 + n + 2)/4

T(n) = O(n2)

1. The student was required to prove that T(n), which is defined by the recurrence relation

T(n) = 2T([n/2]) + 2nlog2n; T(2) = 4

satisfies T(n) = O(nlog2n)

To prove this, the student used the substitution method to prove T(n) = O(nlog2n).

Statement you have to prove:

By using the substitution method, the student will guess T(n) = O(nlog2n) as well as use induction to prove T(n) ≤ cnlog2n

Base Case:

T(2) = 4 ≤ c2log22

4 ≤ c2\*1^2 if c ≥ 2

Inductive Hypothesis:

Now assume T(n/2) ≤ c(n/2)log2(n/2)

Inductive Step:

We must show T(n) ≤ cnlog2n

T(n) ≤ 2(c(n/2)log2(n/2) + 2nlog2n

≤ cnlog2(n/2) + 2nlog2n

Here the student picked an example to help find the constant. By letting n = 8,

You’ll get c4 + 6 ≤ c9 if c ≥ 2

Thus, T(n) ≤ cnlog2n

T(n) = O(nlog2n)

1. *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 *Ω(n2)* which is the lower bound.