Gabriel Emerson (gte0002)

COMP 3270-002

Homework 2

Due 2/16/21

1. Compare
    a. **f(n) = Θ(g(n))**
    b. **f(n) = Θ(g(n))**
    c. **f(n) ∈ Ω(g(n)), g(n) ∈ O(f(n))**
    d. **f(n) = Ω(g(n))**
    e. **f(n) = O(g(n))**
2. Algorithm Mystery
    a. The smallest integer in the array
    b. Base Case: T(1) = 2 (compare i and j, and return it)
    T(n) if(i != j) = 6 (compare i and j, then compute c2, c3, c4, c5, and return)
    [If statement is cost c1, then the rest under the else statement also cost c, this gives us T(c) which is T(n) == 4n+4
    c. –

Level	Level Number	Total # of recursive executions at this level	Input size to each recursive execution	Work done by each execution, excluding recursive calls	Total work done by algorithm
Root	0	0	n/1	1/6	n/6
One level below	1	1	n/2	2/6	2n/6
Two level below	2	2	n/4	4/6	4n/6
One level above base	i-1	N^i/2	n/8	8/6	8n/6
Base case	i	N^i	n/16	16/6	16n/6

    d. Order of complexity = T(n)+2

3. 7T(n/8) + cn; T(1) = c

Level	**Level Number**	**Total # of recursive executions at this level**	**Input size to each recursive execution**	**Work done by each execution, excluding recursive calls**	**Total work done by algorithm**
Root	0	0	Cn	C	Cn
One level below	1	1	7(n/16)	Cn	14(n/16)
Two level below	2	2	7(n/32)	Cn/2	28(n/32)
One level above base	i-1	N^i/2	7(n^2-1)	Cn/n-1	7nlog(n^2-1)
Base case	i	N^I	7(n^2)	Cn/n	7nlog(n^2)

T(n) = 7nlog(n^2)

4. Use substitution method
   Statement of what you have to prove**: T(n) = O(nlogn)**

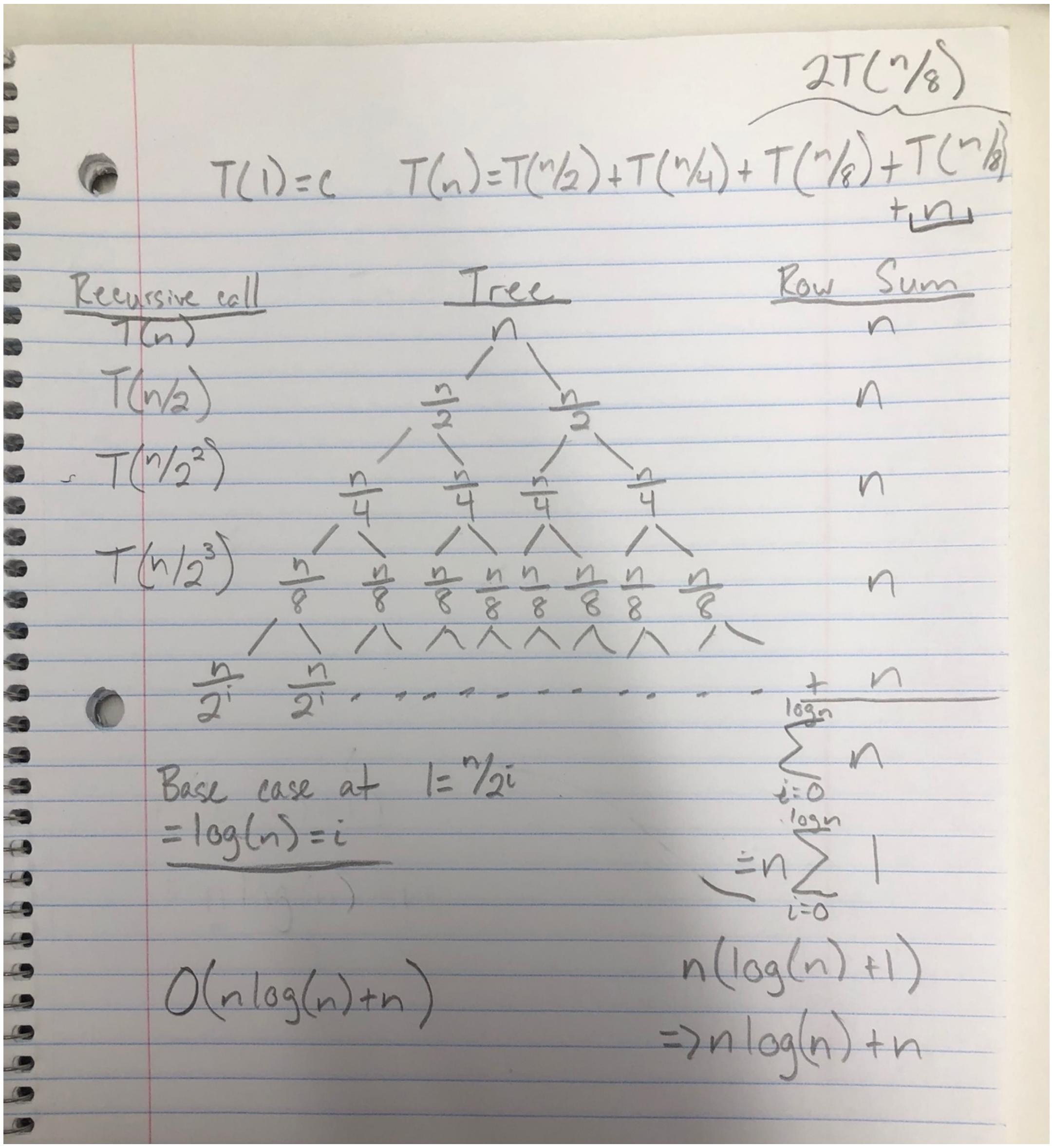
   Base Case proof: **20 = T(3) <= c 3log3 = 0**
   T(3) = 20
   Inductive hypothesis: **T(n) <= cnlog(n)**

   Inductive step: T(n) <= **cnlog(n/3)+5**
   cnlog(n) – cnlog(3) +5
   cnlog(n) – n(c-5) <= nlog(n)

   Value of C: c >= 5

5. Find counterexample: f(n) = O(s(n)) and g(n) = O(r(n)) imply f(n) – g(n) = O(s(n) – r(n))
   If s(n) = n^2 and r(n) = n
   Then f(n) – g(n) = O(n^2) – O(n) == O(n^2)
   But, O(n^2 -n) = O(n)
   O(n^2) != O(n)

6. Guess a plausible solution…:



7. Use substitution to prove previous answer
   Statement what you have to prove: T(n) =T(n/2) +T(n/4) +T(n/8) +T(n/8) +n; T(n) = O(nlog(n) + n)

   Base Case: T(1) = c <= cnlog(n) + n
   Inductive hypopthesis: **T(n) <= cnlog(n)**

   Inductive step: **cnlog(n)+n**
   cnlog(n) +n
   cnlog(n) + n <= nlog(n) + n (if c >= 1)

8. Use Master Method
    a. **A = 2, B = 100/99, f(n) = 100n**
       n^[log_(100/99) 2] ~ n^69
       f(n) = Ω(n^[log_(100/99) 2+ε]
       T(n) = Θ(Ω(n^[log_(100/99) 2+ε]) == Θ(100n)

    b. **A = 16, B = 2, f(n) = n^(3)log(n)**
       n^[log_(2) 16] ~ n^4
       f(n) = Ω(n^[log_(2) 16 +ε]
       T(n) = Θ(n^(3)log(n))

    c. **A = 16, B = 4, f(n) = n^2**
       n^[log_(4) 16] ~ n^2
       f(n) = O(n^(log_4 16)-ε)
       T(n) = Θ(n^(log_4 16))

9. Use backward and forward substitution to solve the recurrence relations
   T(n) = 2T(n-1) + 1
   T(0) = 1
   a. Reverse
      = 2T((n-1)-1) + 1
      = 2T(n-2) + 1 → 2T((n-2)-1) + 1 → 2T((n-3)-1) + 1
      = 1 + 2 + 3 +…
      = (n+1) == T(n)

   Prove T(0) = 1
      T(n) = (n+1)
      T(0) = (0+1)
      = 1
      T(n-1) = (n-1)(n-1+1) is true
   T(n) = T(n-1) + 1
      = (n-1)(n-1+1) + 1
      = (n-1)(n) + 1
      = ((n^2)/2) ((n-n)/2) + 1
      = (n-1) + 1

   b. Forward
      = 2T((n-1)-1) + 1
      = 2T(n-2) + 1 → 2T((n-2)-1) + 1 → 2T((n-3)-1) + 1
      = 1 + 2 + 3 +…
      = (n+1) == T(n)

Prove T(0) = 1

T(n) = (n+1)

T(0) = (0+1)

= 1

T(n-1) = (n-1)(n-1+1) is true

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

= (n-1)(n-1+1) + 1

= (n-1)(n) + 1

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

= (n-1) + 1

c. This equals = **((n-1^(n+1)) – 1)/((n-1) – 1)**

d. **LHS reduces to (n-1) + 1**

RHS reduces by ((n-1^(n+1)) – 1)/((n-1) – 1) → 1^(n+1) = 0

= ((n-(-1) - 1) / (-1)

= (n-1) + 1

e. **This means our complexity is O(n^2)**

10. Solve the recurrence relation

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

T(1) = 1

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

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

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

= 1 + 2 + 3 + … n/2

= n(n+1)/2

Prove T(1) = 1

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

= 1(2)/2

= 2/2

= 1 true

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

T(n-1) + n/2

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

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

= n^2/2 – n/2 + n/2

= n^2/2

= n(n+1)/2

(Big O notation would be O(n^2)

11. Prove that T(n) satisfies T(n) = O(nlog^2(n))
    Since 2T(n/2) goes to floor, we assume that the larger part of this function will be 2nlog^2n, which would satisfy the Big O notation, O(nlog^2n)
    T(n) = 2T(n/2) + 2nlog(n)
    T(2) = 4
    = 2T(n/2) + 2nlog(n)
    = 2T((n/2)/2) + 2(n/2)log(n/2) → 2T(n/4) + (n/2)log(n/2)
    = 2T((n/4)/2) + ((n/2)/2)log((n/2)/2) → 2T(n/8) + (n/4)log(n/4)
    = 2nlogn + logn
    = nlog^2n

12. Problem 3.1-3
    Big O gives the upper bound, or maximum running time of an algorithm. Using the words "at least" here is like abusing the Big O.