Dr. Alice Armstrong

CSC 310

25 February 2021

Algorithms Chapter #2 Homework

***Section 2.1***

1. Indicate natural size, basic operation, and whether the basic operation can be different for inputs of the same size.
   1. Computing the sum of *n* numbers
      1. The natural size metric is *n*.
      2. The basic operation is the addition of two integers.
      3. No, the basic operation count is not different for inputs of the same size.
   2. Computing *n*!
      1. The natural size metric is the magnitude of *n*.
      2. The basic operation is the multiplication of two integers.
      3. No, the basic operation count is not different for inputs of the same size.
   3. Finding the largest element in a list of *n* numbers
      1. The natural size metric is *n*.
      2. The basic operation is the comparison of two numbers.
      3. No, the basic operation count is not different for inputs of the same size.
   4. Euclid’s algorithm
      1. The natural size metric is the magnitude of the largest integer.
      2. The basic operation is modulo division.
      3. Yes, the basic operation count can be different for inputs of the same size.
   5. Sieve of Eratosthenes
      1. The natural size metric is the magnitude of *n*.
      2. The basic operation is removing integers from a list of possible prime integers.
      3. No, the basic operation count is not different for inputs of the same size.
   6. Pen-and pencil algorithm for multiplying two *n*-digit decimal integers.
      1. The natural size metric is *n*.
      2. The basic operation is the multiplication of two digits.
      3. No, the basic operation count is not different for inputs of the same size.
2. What is the basic operation? How many times is it performed as a function of the matrix order *n*? As a function of the total number of elements in the input matrices?
3. Algorithm for adding two *n* x *m* matrices.
   1. The basic operation is the addition of two integers.
   2. It is performed *n*2 times as a function of the matrix order *n*.
   3. It is performed 2*n*2 times as a function of the total number of elements in the input matrices.
4. Algorithm for matrix multiplication.
   1. The basic operation is multiplication because it takes longer on more computers than addition.
   2. It is performed *n* times as a function of the matrix order *n*.
   3. It is performed *n*3 times as a function of the total number of elements in the input matrices.
5. How much a function’s value changes if its argument is increased **fourfold**. Plugging in for all the arguments.
6. Indicate whether the first function has the same, lower, or higher order of growth.
   1. and have the same growth order of .
   2. has a lower order of growth than .
   3. and have the same growth order .
   4. has a higher order of growth than .
   5. and have the same growth order of .
   6. has a lower growth order than .

***Section 2.2***

1. Use the informal definitions of *O*, Θ, and Ω to determine whether the following assertions are true or false. Simplified equation being
   * 1. True
     2. grows no faster than.
     3. True
     4. grows no faster than .
     5. False
     6. on average does not grow at the same rate as .
     7. True
     8. grows at least as fast as .
2. For each of the following functions, indicate the class Θ(g(n)) the function belongs to. (Use the simplest g(n) possible in your answers.) Prove your assertions.
   1. can be reduced to , so . Because the result is > 0, they have the same growth order .
   2. can be reduced to n, so the Because the result is > 0, they have the same growth order .
   3. 2n log(n + 2)2 + (n + 2)2 log n/2 can be reduced to 2n \* 2log(n+2) + (n+2)2 (log n -1) = n log n + n2 log n, so . Because both results are > 0 we pick the one with the highest growth order. So, 2n log(n + 2)2 + (n + 2)2 log n/2 .
3. List the following functions according to their growth from lowest to highest
   1. (n-2)!, 5log(n+100)10, 22n, 0.001n4 + 3n3 + 1, ln2n, , 3n
   2. (n-2)! ∈ Θ(n!), 5log(n+100)10 ∈ Θ(log(n)), 22n ∈ Θ(4n), 0.001n4 + 3n3 + 1 ∈ Θ(n4), ln2n ∈ Θ(log2n), ∈ Θ(n1/3), 3n ∈ Θ(3n)
   3. Θ(log(n)) < Θ(log2n) < Θ(n1/3) < Θ(n4) < Θ(3n) < Θ(4n) < Θ(n!)
   4. 5log(n+100)10 < ln2n < < 0.001n4 + 3n3 + 1 < 3n < 22n < (n-2)!
4. Prove the sections theorem for:
   1. notation

t(n) is said to be in (g(n)), if t(n) is bounded below some positive constant multiple of g(n) for all large n

n3 c \* n2

For some c = 1 and n0 = 0:

| n | n3 | 1\*n2 |
| --- | --- | --- |
| 1 | 1 | 1 |
| 2 | 8 | 4 |
| 3 | 27 | 9 |
| 4 | 64 | 16 |
| 5 | 125 | 25 |
| 6 | 216 | 36 |

***Section 2.3***

1. Compute the following sums:
   1. 1 + 3 + 5 + 7 + … + 999 =
   2. (n+1) - 3 + 1 = n-1

**g.**

1. Find the growth order of the following sums:
   1. ∈ Θ(n\*(n2 + 1)2) = Θ(n\*n4) + Θ(n\*2n2)+ Θ(n\*1) = Θ(n5) ~~+ Θ(2n~~~~3~~~~)~~~~+ Θ(n)~~ = Θ(n5).
   2. = = 2 \*∈ Θ(~~2\*~~(n-2)\*log(n)) = Θ(n log(n)) ~~- 2 \* Θ( log(n))~~ = Θ(n log(n)).
2. The sample variance of *n* measurements x1, x2, …, xn can be computed as   
   Divisions: Assuming x bar is only calculated once- 2 (/n once and /n-1 once)  
   Multiplications: n ( is multiplied by itself n times)  
   Additions / Subtractions:   
   A: Sum from 1 -> n -1 for 2 sequences of sums, so additions are done 2(n-1) times.   
   S: Subtraction is done in the sequence from 1-> n-1 and once afterwards in the dividend, so subtractions are done (n-1) + 1 times.  
   A/S: 2(n-1) + (n-1)+1 = 2n - 2 + n = 3n - 2

Divisions: 2 ( /n once and /n-1 once)

Multiplications: n + 1 (first sum sequence multiplies x, n times; second sum sequence is multiplied by itself once after summing from 1 to n)

Additions / Subtractions:

A: Sum from 1 to n for 2 sequences of sums, so additions are done 2(n) times.

S: Subtraction is done after the second sequence of sums is completed and squared and once after in the dividend, so subtractions are done 2 times.

A/S: 2n + 2

***Section 2.4***

1. Solve the following recurrence relations.
   1. for

* 1. for

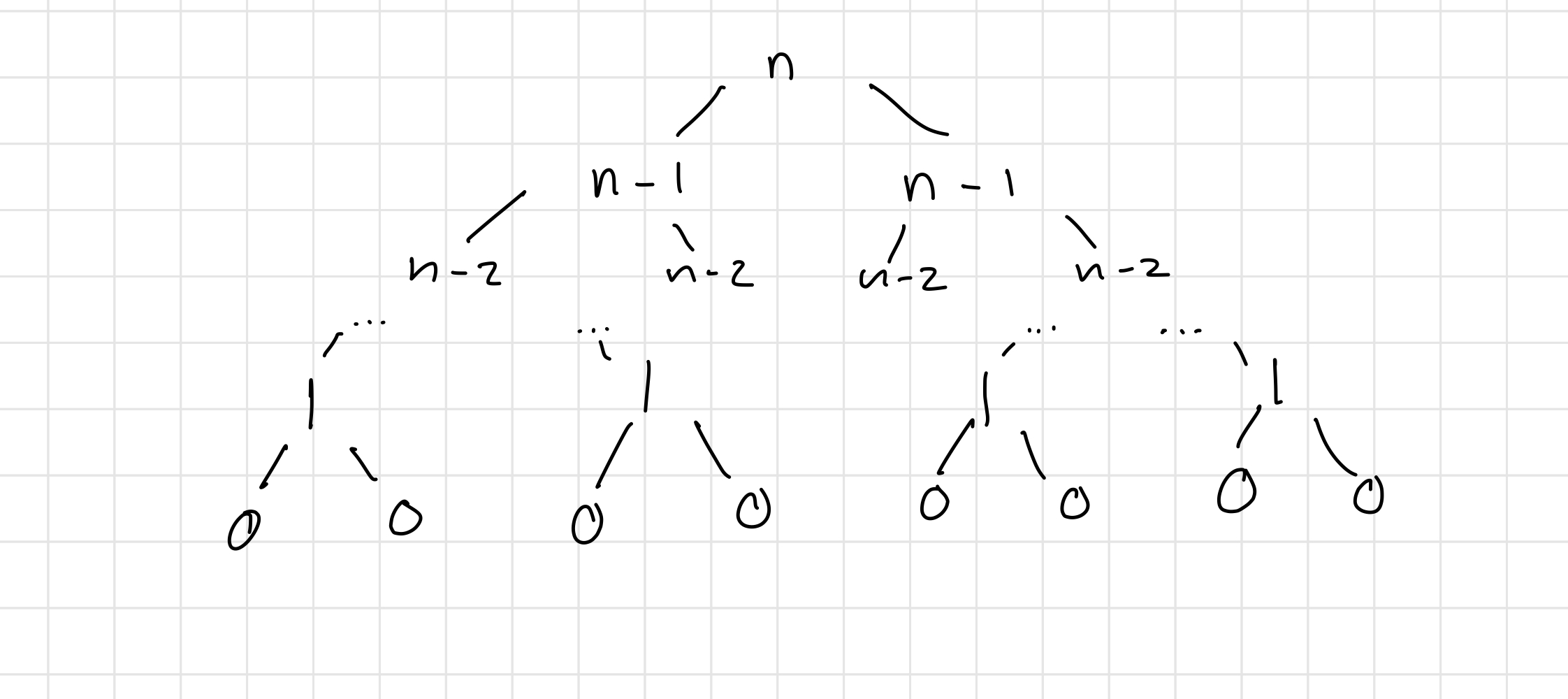
=

=

=

=

* 1. for

1. Consider the following recursive algorithm for computing the sum of the first *n* cubes:  
      
     
   **Algorithm** *S(n):*  
   // Input: A positive integer *n.*  
   // Output: The sum of the first *n* cubes.  
     
     
   1. The basic operation for this algorithm is multiplication and it occurs 2 times per loop. So if C(n) represents the number of time it occurs it would be . The pattern continues as followed. , . A pattern of hold. Since we know C(1) = 0, we can say that C(n) = C(1) + 2(n - 1) = 2(n -1).
   2. A non-recursive alternative to S(*n*) **Algorithm** NRS(*n*):  
      // Input: A positive integer *n.*  
      // Output: The sum of the first *n* cubes.
2. Design a recursive algorithm for computing 2^n for any nonnegative integer *n* that is based on the formula .
   1. **Algorithm** *TwoToThePowerOf(n):*// Compute using the formula .  
      // Input: Any nonnegative integer *n.*// Output: Return .
   2. The basic operation for this algorithm is addition so if A(n) represents the number of times addition occurs it would be . The pattern continues as followed. , . The pattern of holds. Since we know that , we can say that .
   3. 
   4. This is a bad algorithm because it requires several recursive function calls that repeat the same calculation when you could just do something like a for/while loop to non-recursively calculate the number.