Brycen Martin



LM2 Written

1. A.) x(n) = x(n-1) + 5 for n > 1, x(1) = 0

x(1) = 0

x(2) = 0 + 5 = 5



x(3) = 5 + 5 = 10

x(4) = 10 +5 = 15

**x(n) = (n\*5) – 5**

B.) x(n) = 3x(n-1) for n > 1, x(1) = 4



x(1) = 4



x(2) = 3(2)(1) = 6

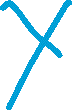
x(3) = 3(3)(2) = 18



x(4) = 3(4)(3) = 36



x(5) = 3(5)(4) = 60



**x(n) = 4-(6x/(x-1)^3)**

E.)

x(n) = x(n/3) + 1 for n > 1 , x(1) = 1 solve for n = 3^k

x(1) = 1



x(2) = 2(0.66)+1 = 2.32



x(3) = 3(1)+1 = 4



x(4) = 4(1.33)+1 = 6.32



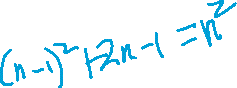
x(5) = 5(1.66)+1 = 9.3

x(6) = 5(2)+1 = 11

**x(3^k) = 3^k((3^k)/3) + 1 = 3^(2k-1) + 1**

1. A.) x(n) = x(n-1) + 2\*n -1 for n > 1, x(1) = 1

x(1) = 1



x(2) = 2(1) + 2\*1 = 4



x(3) = 3(2) + 2\*2 = 10

x(4) = 4(3) + 2\*3 = 18

x(5) = 5(4) + 2\*4 = 28

x(6) = 6(5) + 2\*5 = 40

x(n) = ((2(x-2)(x))/((x-1)^3)) + 1

B.) The recurrence relation should be identical to part A due to the fact that they are both run the same amount of times, so the algorithm would be ((2(x-2)(x))/((x-1)^3)) +1



C.) the number of subtractions and additions made by the algorithm is just 3 times the number of multiplications according to the formula we were given so the formula should be



3x^3–7x^2+5x– 3/(x-1)^3



10.) I think the worst case efficiency for this algorithm would be n^2 due to the fact the algorithm does a search from 0 to n in the second if statement and then has to check if array[n-1 ,j] = 0 which it will have to do n times so n times by n times is n^2.

