# Q1: DPV .3a

Prove Fn ≥ 2.5n for n ≥ 6.

First I will adjust the initial statement with strong induction:

P(n) = Fn ≥ 2.5n and Fn+1 ≥ 2.5(n+1)

I will prove P(n) for n ≥ 6 with induction which in turn proves Fn ≥ 2.5n for n ≥ 6.

First, the base case.

P(6): F6 ≥ 2.5\*6 and F7 ≥ 2.5\*7

The sixth Fibonacci number is 8. 23 = 8. The first part of the statement checks out.

The seventh Fibonacci number is 13. 23.5 = 11.31. The second part of the statement checks out.

Now the induction step.

I will prove P(n) = Fn ≥ 2.5n and Fn+1 ≥ 2.5(n+1) 🡪 P(n+1) = Fn+1 ≥ 2.5(n+1) and Fn+2 ≥ 2.5(n+2)

I will assume P(n) to be true and use it to prove P(n+1).

The first statement in P(n+1), Fn+1 ≥ 2.5(n+1) we know is true because it is an assumption made via the induction hypothesis.

The second statement in P(n+1), Fn+2 ≥ 2.5(n+2) can be proven like so.

Fn+2 ≥ Fn+1 + Fn (this uses the properties of the Fibonacci series)

Fn+2 ≥ 2Fn (since we know the Fibonacci series is increasing, 2Fn is less than Fn+1 + Fn)

2.5(n+2) ≥ 21\*2.5n (I just subbed in the exponential representations of the numbers)

2.5n \* 21 = 21\*2.5n (It becomes clear that the two equations equal each other)

That proves that Fn+2 ≥ 2.5(n+2), which proves P(n) 🡪 P(n+1), which proves P(n) is true for all n greater than 6, which proves the initial statement of Fn ≥ 2.5n for n ≥ 6.

# Q1: DPV .3c

Following the same setup as before: Fn ≥ 2cn and Fn+1 ≥ 2c(n+1) 🡪 Fn+2 ≥ 2c(n+2)

Solve for the highest value of C possible.

Fn+2 ≥ Fn+1 + Fn  (Using the properties of the Fibonacci series)

2c(n+2) ≥ 2c(n+1) + 2cn (Subbing in the exponential forms)

2cn \* 22c ≥ 2cn \* 2c + 2cn (Using the properties of exponents and common bases)

2cn \* 22c ≥ 2cn (2c + 1) (Using the distributive property)

22c ≥ 2c + 1 (Canceling out terms)

u2 ≥ u + 1 (subbing in u for 2c)

u2 – u – 1 ≥ 0 (rearranging function)

u = 2c = (quadratic formula)

# Q2

I typed up this short Python program to mimic the function bar:

def bar(n):

print("\*", end="")

if n==0:

return

else:

for i in range(0,n):

bar(i)

return

if \_\_name\_\_=='\_\_main\_\_':

for nums in range(0,10):

print("n=",nums," ",end="")

bar(nums)

print("")

I counted the number of stars in each row of output to construct this table.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| n | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| # stars | 1 | 2 | 4 | 8 | 16 | 32 | 64 | 128 | 256 | 512 |

For here, it’s obvious to see that the function T(n) = 2n gives the number of stars printed as a function of n.

One can also write the function, bar(n), out recursively as bar(n) = 1 + bar(0) + bar(1) + bar(2) + … + bar(n-1)

I’ll prove that T(n) accurately models that bar function with structural induction – proving that the bar function will double the value of the previous result and that this property is preserved.

The base case bar(0) = 1 star

The constructor rule / property of the function is that bar(n) = 2bar(n-1). Since the lowest value is 1, and each value will be doubled, all successive values will be multiples of 2. This is consistent with the definition of T(n) = 2^n, so the functions are the same.

# Q3:a-f

1. The limit is a constant, so the first (top) function is Θ (has the same growth rate) of the bottom function
2. Thus, 100n2 is O(.01n3)
3. . Thus, the first function is Θ of the bottom function
4. The top divided by the bottom goes to infinity as n approaches infinity. Thus, the top function is Ω of the bottom function.
5. The top divided by the bottom approaches a constant for the first / top function is Θ of the second / bottom function.
6. The first / top function is order O of the second / bottom function.

# Q4

Order of the functions from least to greatest in terms of growth rates: