Aaron Floreani

Homework 4

CMSI-2120-02

1. Write 𝒫({𝑎,𝑏,𝑐})P({a,b,c}) in a form in which all elements are listed.

𝒫 { {}, {a}, {b}, {c}, {a, b}, {a, c}, {b, c}, {a, b, c} }

2. How many partitions are there of {𝑥∈ℕ∣1≤𝑥∧𝑥≤10}{x∈N∣1≤x∧x≤10}?

According to the 10th number in the Bell’s number list: 1, 2, 5, 15, 52, 203, 877, 4140, 21147, 115975. There are 115975 ways to partition B(10).

3. Let 𝐴 be a set and 𝑚 and 𝑛 be positive integers. Would you say that 𝐴^(𝑚+𝑛)=𝐴^𝑚×𝐴^𝑛? Give arguments for or against accepting this equality as fact. (Hint: you may want to consider the sets 𝐴×𝐴2 and 𝐴2×𝐴.)

If the same operations that apply to numbers apply to sets, then multiplying exponents with the same base is equal to that base with the numbers added. Therefore assuming numbers function the same as sets, then the equation 𝐴^(𝑚+𝑛)=𝐴^𝑚×𝐴^𝑛 holds true.

4. Express, in lambda notation, the function which when passed two functions 𝑓 and 𝑔, returns the composition of 𝑓 and 𝑔. (Don't use the predefined ∘ symbol for composition. Actually define the meaning of composition in your answer.

The composition of two functions g and f is the new function we get by performing f first, and then performing g. Function composition is applying one function to the results of another: 𝜆(𝑔,𝑓).𝜆𝑥.𝑓(𝑔(𝑥))

5. Let 𝑓=𝜆𝑥.4𝑥−3𝑥2.

𝑓0(0.01) = 0.01

𝑓1(0.01) = 0.0397

𝑓2(0.01) = 𝑓1(0.0397) = 0.15407173

𝑓5(0.01) = 𝑓1(1.28897800119) = 0.1715191421

𝑓30(0.01) = 0.3741465082616884

𝑓50(0.01), according to Java = 1.3084580315967430

𝑓50(0.01), according to your handheld calculator =

Not quite sure but around: 1.3084580315967430.

Explain why we can never write out the exact value, as a decimal number, of 𝑓75(0.01) in 10 point Times New Roman on printer paper that is produced on Earth.

The decimal number for 𝑓75(0.01) will be too many digits to fit on all the printer paper produced on earth.

Do you think it will possible in our lifetime whether we will ever know the first digit of the decimal expansion of 𝑓100(0.01). Why or why not?

I believe we can, according to Java the first digits are 0.1521912397012866. However, Java is known to be wacky with numbers and has low precision for accuracy. However, I still believe the first digit produced by Java may be reasonably accurate and is most likely 1.

6. What is the time complexity of this code fragment? (Use Θ-notation)

At n= 10 println is called 1024 times and at n = 5, println is called 32 times. Continuing this, we can deduce the time complexity is Θ(2^n).

7. What is the time complexity of this code fragment? (Use Θ-notation)

The code runs forever, so the time complexity is Θ(∞).

8. What is the time complexity of this code fragment? (Use Θ-notation)

The outer loop runs i^2 = n times, which is √n times. The inner loop runs 2^i = n times, which we can take the log of to get log(n). (The log is base 2). Time complexity is Θ(√𝑛 \* log n).

9. What is the time complexity of this code fragment? (Use Θ-notation)

When n = 8, 15 stars are printed. For n= 16, 31 stars are printed. For n = 64, 127 stars are printed. Therefore T(n) = 2n -1, which is Θ(n)

10. What is the time complexity of this code fragment? (Use Θ-notation)

The outer loops runs n^2 times, and the inner loop runs j^2 = n times, which is log(n) times. Therefore Θ((n^2) \* log(n))

11. What is the time complexity of this code fragment? (Use Θ-notation)

The outer loop runs n times. The inner loop runs 2^j = i times, which is log(i) times. Maybe we can set n = i and guess that the complexity is Θ(n \* log(n))?

12. What is the time complexity of this code fragment? (Use Θ-notation)

The outer loop runs n/4 times and the inner loop runs n times. Therefore the time complexity is Θ(n^2).

13. Give both the best-case and worst-case time complexities of this code fragment. (Use Θ-notation)

The best case scenario is having the inner loop not run, and only the outer loop runs. The outer loop runs log(n) times, which is Θ( logn ). The worst case is when the outer loop and inner loops both run, which is Θ( n \*log(n) ) because the inner loop runs i times, which is equal to n.

14. Give both the best-case and worst-case time complexities of this function. (Use Θ-notation)

The best case is that n is equal to 0 and therefore the function is performed in linear time, where Θ( n ). The worst case is that the function is called recursively c (a constant) amount of times. This case would also lead to Θ( n ).

15. What is the time complexity (in Θ-notation) of a procedure to print out the exact value of 2^𝑛, where 𝑛 is a nonnegative (big) integer? The procedure described is supposed to print a result no matter how large the result may be.

Since this algorithm is completed in one step which is 2^n, and therefore the time complexity is Θ( 2^n ).

16. An algorithm with time complexity 𝑇(𝑛)=𝑛^3 can process a 100-element list on our PC in 10 seconds.

a. How long would it take to process a 200-element list?

First we calculate the speed of our computer: 100^3 operations / 10 seconds = 100000 operations per second.

Next we calculate the time for a 200 element list:

(200^3 operations) / 100000 operations/seconds = 80 seconds.

b. If we ran the algorithm on a machine that was 10 times faster than our PC, how large of a list could we process in 30 seconds?

Instead of 100000 operations per second, this machine does 1000000 operations per second. Therefore (n^3) / 1000000 = 30.

(n^3) = 30000000

Therefore n = 310.723250595

c. How much faster than our PC would a computer have to be in order to process a 1000000000-element list in a time span of 1 hour?

(1000000000^3 operations) / x = 216000 seconds

x = 4.6296296e+21

4.6296296e+21 = 100000 \* y

4.6296296e+16 times faster of a computer is needed.

17. An algorithm with complexity function 𝑇(𝑛)=𝑛log𝑛 processes a 64-element list in three minutes and 12 seconds on our PC.

a. How long does it take to process a 128-element list?

First we calculate the speed of the PC:

(log(64) × 64) operations / 192 seconds

= 2 operations per second. (Assuming log is base 2).

b. How large of a list could a computer that is 8 times faster than our PC process in 10 seconds?

2 \* 8 = 16 operations per second

(n \* log(n) ) / 16 = 10 seconds

(n \* log(n) ) = 160

n = 32

c. How much faster would a computer have to be than our PC to process a list of size 10 in a second?

(10 log(10) ) / x = 1

x = 33.2193

33.2193 = 2 \* y

The computer would have to be 16.60965 times faster than our current PC.

18. An algorithm with time complexity function 𝑇(𝑛)=2𝑛log𝑛 can process a 32 element list in 2 minutes and 40 seconds on our PC.

a. How long would it take to process a 64 element list?

First we calculate the speed of our PC (2 \* 32 \* log(32)) / 160 seconds = 2 operations per second.

(2 × 64 × log(64)) = 768

768 / 2 = 384 seconds

b. If we ran the algorithm on a machine that was 4 times faster than our PC, how large of a list could we process in 16 seconds?

The speed of this PC is: 2 \* 4 = 8 operations per second.

2nlog(n) / 8 = 16

2nlog(n) = 128.

n = 16

19. We have seen that there is little hope of solving problems of size 100 or so with algorithms of complexity𝜆𝑛.2𝑛 even when billions of operations can be carried out per second. But what about algorithms of complexity 𝜆𝑛.1.1𝑛? How do these algorithms compete with quadratic algorithms? In particular, if a billion operations can be performed in one second, up to what problem size will the 𝜆𝑛.1.1𝑛 be faster than a 𝜆𝑛.2𝑛 algorithm?

1.1^n will be slower than n^2 when 1.1^n = n^2 which is when n is above 95.7168. Exponential complexity will always eventually be slower than quadratic complexity.

20. Rank each of the following functions by growth rate: 𝜆𝑛.𝑛, 𝜆𝑛.𝑛2, 𝜆𝑛.𝑛1.5, 𝜆𝑛.𝑛√, 𝜆𝑛.𝑛log𝑛, 𝜆𝑛.𝑛loglog𝑛, 𝜆𝑛.𝑛2log𝑛, 𝜆𝑛.𝑛log𝑛2, 𝜆𝑛.𝑛(log𝑛)2, 𝜆𝑛.2, 𝜆𝑛.𝑛3, 𝜆𝑛.2𝑛, 𝜆𝑛.2log𝑛, 𝜆𝑛.2𝑛, 𝜆𝑛.2log𝑛, 𝜆𝑛.2𝑛⋅𝑛, 𝜆𝑛.2𝑛÷2, 𝜆𝑛.𝑛!, 𝜆𝑛.𝑛𝑛, 𝜆𝑛.(log𝑛)𝑛, 𝜆𝑛.log𝑛𝑛, 𝜆𝑛.log(𝑛√).

Lowest to highest growth rate list:

𝜆𝑛.(2/𝑛)

𝜆𝑛.2

𝜆𝑛.log√𝑛

𝜆𝑛.2log𝑛

𝜆𝑛.√𝑛

𝜆𝑛.𝑛

𝜆𝑛.2^(log𝑛)

𝜆𝑛.𝑛loglog𝑛

𝜆𝑛.𝑛log𝑛

𝜆𝑛.log𝑛𝑛

𝜆𝑛.𝑛log(𝑛^2)

𝜆𝑛.𝑛(log𝑛)^2

𝜆𝑛.𝑛^(1.5)

𝜆𝑛.𝑛^2

𝜆𝑛.(𝑛^2)log𝑛

𝜆𝑛.𝑛^3

𝜆𝑛.2^(𝑛÷2)

𝜆𝑛.2^𝑛

𝜆𝑛.(log𝑛)^𝑛

𝜆𝑛.𝑛!

𝜆𝑛.𝑛^𝑛

𝜆𝑛.2^(𝑛⋅𝑛)