CMSI 2120: Data Structures
HW #4

[Christina Choi]

November 21, 2022

Q1

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

$\{\{a\},\{b\},\{c\},\{a,b\}, \{a,c\},\{b,c\},\{a,b,c\},\{\}\}$

Q2

How many partitions are there of $\{x \in \mathbb{N} | 1 \leq x \land x \leq 10\}$?

115975. This is the Bell Number, which counts the possible partitions of a set. In this case, in a set of 10, the number of possible partitions is 115,975.

Q3

Let A be a set and m and n be positive integers. Would you say that $A^{m+n} = A^m$ x A^n? Give arguments for or against accepting this equality as fact. (Hint: you may want to consider the sets AxA^2 and A^2xA)

No, $A^{m+n} = A^m$ x A^n. When considering the sets AxA^2 and A^2xA, the Cartesian product of two sets is distributing the first set to all elements of the second set, putting the element of the first set first. For example, $A = \{a, b\}$. $A^2 = \{(a, a), (a, b), (b, a), (b, b)\}$. Now, $AxA^2 = \{(a, a, a), (a, a, b), (a, b, a), (a, b, b), (b, a, a), (b, a, b), (b, b, a), (b, b, b)\}$. And $A^2xA = \{(a, a, a), (a, b, b), (b, a, a), (b, b, a), (a, a, b), (a, b, b), (b, a, b), (b, b, b)\}$. Even though they are mixed up, because we are doing the cartesian product of identical sets, in the end, they end up becoming the same set. If they were different sets, we would not be able to say yes.

Q4

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 $\circ$ symbol for composition. Actually define the meaning of composition in your answer.

$\lambda(f, g).\lambda x.g(f(x))$

Q5

Let $f = \lambda x.4x - 3x^2$

(a) $f^0(0.01) = 0.01$

(b) $f^1(0.01) = 0.0397$

(c) $f^2(0.01) = 0.15407173$

(d) $f^5(0.01) = 0.1715191421091756109132737611936695015318486615246297164053309653$

(e) $f^{30}(0.01) =$ We cannot be sure of this answer using just a calculator because the answers will be rounded at each term, but we do know that the number of digits can be determined by 2^{n+1}, so f^30 will have $2^{30+1} = 2147483648$ digits.

(f) $f^{50}(0.01)$ according to Java $= 1.3056$

(g) $f^{50}(0.01)$ according to your hand-held calculator $= 0.0892$ (very different from Java)

(h) Explain why we can never write out the exact value, as a decimal number, of $f^{75}(0.01)$ in 10 point Times New Roman on printer paper that is produced on Earth.

The full decimal number will be way too big to fit on any size paper. Solving out f^75 will have 75,557,863,725,914,323,419,136 digits after the decimal point. And that number will be far too large to fit on any paper on Earth.

(i) Do you think it will possible in our lifetime whether we will ever know the first digit of the decimal expansion of $f^{100}(0.01)$. Why or why not?

No, it will not be possible. With each iteration of this function, there is a certain amount of rounding because the decimal places get increasingly very high. Therefore, once we reach the 100th iteration, it is difficult to know for certain what the true first digit will be without rounding.

Q6

What is the time complexity of this code fragment? (Use θ-notation)

$\theta(2^n)$
The function x = x + x in the outer loop leads to the value of x being: 1, 2, 4, 8... This is the same as 2^x. In the inner loop, the variable j prints "*" x times. Therefore, the time complexity will be 2^n.

Q7

What is the time complexity of this code fragment? (Use θ-notation)

$\theta(\infty)$
In the inner loop, the value of variable j is never changing, as variable i is doubling everytime, so we never escape the inner loop and will print "*" for infinity.

Q8

What is the time complexity of this code fragment? (Use θ-notation)

$\theta(\sqrt{n} * log(n))$
In the outer loop, we are performing $i^2 = n$ times, meaning $\sqrt{n}$ times. In the inner loop, we are performing log(n) times.

Q9

What is the time complexity of this code fragment? (Use θ-notation)

$\theta(n)$
This code prints 1+4+8+16...n stars. The equation for this will be 2n-1, so the time complexity for this code will be n.

Q10

What is the time complexity of this code fragment? (Use θ-notation)

$\theta(n^2 * log(n))$
In the outer loop, we are performing n^2 times, and in the inner loop, we are performing log(n) times.

Q11

What is the time complexity of this code fragment? (Use θ-notation)

$\theta(n * log(n))$
In the outer loop, we are performing n times, and in the inner loop, we are performing log(n) times.

Q12

What is the time complexity of this code fragment? (Use θ-notation)

$\theta(log(n) * n)$
In the outer loop, we are performing log(n) times. And in the inner loop, we are performing n times.

Q13

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

The time complexity of this program is $\theta(log(n) * n)$. The worst-case time complexity for this program is $\theta(n)$. The best-case time complexity for this program is $log(n)$.
The outer loop will always perform $log(n)$ times in both best and worst cases. In worst case scenario, Instant.now() will always be greater than t (we never break out of the inner loop)and the inner loop will perform n times. Therefore, the time complexity becomes $\theta(n)$. In best case scenario, Instant.now() is never greater than t(less than or equal to) and the inner loop will perform 0 times.

Q14

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

What this code does: Does n = 0? If yes, return 1. If not, do the $n\%2 = 0$ equation. Does n mod 2 = 0? If yes, do power(x*x, n/2). If not, do x*power(x*x, n/2).
In the best case scenario, $n\%2 = 0$ and we do power(x*x, n/2), which performs log(n) times. In the worst case scenario, $n \div 2 \neq 0$ and we do x*power(x*x, n/2), which also performs log(n) times. Therefore, in both the best and worst case scenario, the time complexities will be the same.

Q15

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

To print out the exact value of 2^n, I would create a program with a for-loop that performs n times, in which we are adding 2 n times.(2, 2+2, 2+2+2...) When the input has n bits, the output will have 2^n bits. Therefore, the time complexity will be 2^n.

Q16

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?
For a 200-element list, the total number of instructions will be: $T(200) = (200)^3 = 8000000 = 8x10^6$
Looking at the given information, the total number of instructions for a 100-element list will be $T(100) = (100)^3 = 1000000 = 10^6$. Processing a 100-element list on our PC in 10 seconds means $10^6/10 = 10^5$ instructions per second.
Therefore, processing a 200-element list on our PC will take $(8x10^6)/(10^5) = 80$ 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?
If a machine is 10 times faster than our PC, it will process $10 * 10^5 = 10^6$ instructions per second instead. Therefore, in 30 seconds, that will be $30 * 10^6$ instructions. Then to find how large that list will be: $(30 * 10^6)^{1/3} = 310.723$ 310 elements.

(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?
One hour contains 3600 seconds.
This list contains 10^9 elements, which equates to 10^{27} instructions when we cube it.
Transferring the given information of 100 elements on a PC to hours will give us $3600 * 10^5 = 36 * 10^7$ instructions processed in one hour.
We will need a machine that is $10^{27}/(36 * 10^7) = 2.778 * 10^{18}$ times faster than our PC.

Q17

An algorithm with complexity function $T(n) = n * log(n)$ 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?
Looking at the given information, a 64 element list is $64 * log(64) = 384$ instructions. Processing 384 instructions in 192 seconds(3 minutes and 12 seconds) gives us $384/192 = 2$ instructions per second. Using this information for a 128 element list, we can see that 128 elements is $128 * log(128) = 896$. To process 896 instructions will take $896/2 = 448$ seconds.

(b) How large of a list could a computer that is 8 times faster than our PC process in 10 seconds?
If a machine ran 8 times faster than our PC, that means it will process $8 * 2 = 16$ instructions per second. And in 10 seconds, that is $16 * 10 = 160$ instructions. To find the number of elements in this list, $n * log(n) = 160$, which gives us 32, approximately 32 elements in this list.

(c) How much faster would a computer have to be than our PC to process a list of size 10 in a second?
A list of size 10 will have $10 * log(10) = 33.22$ instructions.
We will need a machine that is $33.22/2 = 16.61$ times faster than our PC.

Q18

An algorithm with time complexity function $T(n) = 2n * log(n)$ 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?
Using the given information, a 32 element list will be $2(32) * log(32) = 320$ instructions. Processing 320 instructions in 160 seconds(2 minutes and 40 seconds) will give us $320/160 = 2$ instructions per second.
A 64 element list will be $2(64) * log(64) = 768$ instructions. That means it will take $768/2 = 384$ seconds to process.

(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?
A machine that is 4 times faster than our PC would be $4 * 2 = 8$ instructions per second.
That will be $8 * 16 = 128$ instructions in 16 seconds.
To find the number of elements in this list, $2n * log(n) = 128$, which gives us 16 elements.

Q19

We have seen that there is little hope of solving problems of size 100 or so with algorithms of complexity $\lambda n.2^n$ even when billions of operations can be carried out per second. But what about algorithms of complexity $\lambda n.1.1^n$? 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 $\lambda n.1.1^n$? be faster than a $\lambda n.n^2$ algorithm?

The growth rate of $\lambda n.1.1^n$ is significantly lesser than the growth rate of $\lambda n.2^n$, so it would be more feasible to solve problems of size 100. And $\lambda n.n^2$ would even easier to solve. With a machine that can perform a billion operations per second, $\lambda n.1.1^n$ would be faster than $\lambda n.n^2$ up to 95.72 elements, roughly 95 elements.

Q20

Rank each of the following functions by growth rate: $\lambda n.n$, $\lambda n.n^2$, $\lambda n.n^{1.5}$, $\lambda n.n^{0.5}$, $\lambda n.nlogn$, $\lambda n.nloglogn$, $\lambda n.n^2logn$, $\lambda n.nlog(n^2)$, $\lambda n.n(logn)^2$, $\lambda n.2$, $\lambda n.n^3$, $\lambda n.2/n$, $\lambda n.2logn$, $\lambda n.2^n$, $\lambda n.2^{logn}$, $\lambda n.2^{n^2}$, $\lambda n.2^{n/2}$, $\lambda n.n!$, $\lambda n.n^n$, $\lambda n.(logn)^n$, $\lambda n.log(n^n)$, $\lambda n.log(\sqrt{n})$.

$\lambda n.2/n$
$\lambda n.2$
$\lambda n.log(n^{0.5})$
$\lambda n.2log(n)$
$\lambda n.n^{0.5}$
$\lambda n.n$
$\lambda n.2^{logn}$
$\lambda n.nloglogn$
$\lambda n.log(n^n)$
$\lambda n.nlogn$
$\lambda n.nlog(n^2)$
$\lambda n.n^{1.5}$
$\lambda n.n(logn)^2$
$\lambda n.n^2$
$\lambda n.n^2logn$
$\lambda n.n^3$
$\lambda n.2^{n/2}$
$\lambda n.2^n$
$\lambda n.(log(n))^n$
$\lambda n.n!$
$\lambda n.n^n$
$\lambda n.2^{n^2}$