CSCI 3104 Summer 2018
Problem Set 2
Eischen, Parker
10/31
parker.eischen@colorado.edu

1. (10 points) For each of the following claims, determine whether they are true or false. Justify your determination (show your work). If the claim is false, state the correct asymptotic relationship as O, Θ, or Ω. Unless otherwise specified, lg is log2.

(a) n + 1 = O(n^4)
n +1 <= c(n^4)
c can be 2 for example and this will be true
TRUE
(b) 2^(2n) = O(2^n)
2^(2n) <= c2^n
ln(2) * 2n <= ln(c) + ln(2)*n
2n<= ln(c) + n
n<=ln(c)
No possible value of c completely satisfies this equation
FALSE
(c) 2^n = Θ(2^(n+7))
lg(2^n) = lg(2^(n+7))
n=n+7
c1(n+7)<=n<=c2(n+7)
C1 cannot be any value > 1 where this is true, so that means this equation is
FALSE
(d) 1 = O(1/n)
1 <= (1/n)
n<=1
As n increases it becomes greater than 1 so this inequality becomes false
FALSE
(e) ln^2 n = Θ(lg^2n)
c1(lg^2(n))<=ln^2 (n) <=c2(lg^2(n))
lg(n) > ln(n) for all n >1
The upper bound of this equation is satisfied, however the for the lower bound, there is no c1 that can satisfy
FALSE
(f) n^2 + 2n − 4 = Ω(n^2)
n^2 + 2n - 4 >= n^2 both go to inf so we can use L'Hospital's
lim ->(2n+2/2n) again
lim -> (2/2) = 1

Since the limit = 1 then the result is considered Θ(n^2)
FALSE
(g) 3^(3n) = Θ(9^n)
c1(9^n) <= 3^(3n) <= c2(9^n)
c1(9^n) <= 3^(3n)
log3(c1*9^n) <= log3(3^(3n))
n*log3(c1*9) <= 3n*log3(3)
n*log3(c1*9) <= 3n
If c1 = 1 n*log3(9) <= 3n
n*2 <=3n
This side of the inequality works
Simplifying a bit
3n <= n * log3(c2*9)
If c2 = 9
3n <= n * log3(81)
3n<= n * 4
Works out
TRUE
(h) 2^(n+1) = Θ(2^(n lg n))
c1*2^(nlgn) <= 2^n+1 <= c2*2^(nlgn)
c1*lg(2^nlgn) <= lg(2^(n+1))
C1nlgn <= n+1
If c1 = 1 this is true
n+1<= c2*nlg(n)
If c2 =100, 100nlg(n) is still less than n+1 when n = 1. So this statement is false for all values of c2. Meaning this statement is
FALSE
(i) √ n = O(lg n)
*√ n <= c*lg(n)*
lg(n) < n for n> 0
lg(n) = 2lg(√ n*) < 2√ n*
Which means that lg(n) = O(√ n) this cannot be flipped to satisfy the other way around
FALSE
(j) 10^100 = Θ(1)
10^100 is a constant, this value will not change as the value of n increases. Since it is a constant, it will have the constant asymptotic relationship which is Θ(1c). Where c = 10^100. This simplifies down to Θ(1)
TRUE

2. (30 points) Shadow the Rogue is running a con in town after returning from the ancient dwarven mine. She gives a victim 12 cups and a handful of gold and tells them to place some amount of gold under each cup. One by one, the victim reveals the gold under each cup in turn

and then puts the cup down, and Shadow has to then choose which two cups have the largest difference in gold. If Shadow is correct, the victim pays 5 gold, otherwise the victim receives all the gold under the cups. Fortunately, Shadow has a photographic memory and can remember the sequence in her head.
For example, let Cups = [8, 9, 3, 4, 14, 12, 15, 19, 7, 8, 12, 11]. If we choose cup i = 2 with Cups[i] = 3 and j = 7 with Cups[j] = 19, the difference is 19 − 3 = 16 gold.

(a) Consider the pseudocode below that takes as input an array A of size n:

```
conningTheTownspeople(A) :
    maxGoldSoFar = 0
    for i = 0 to length(A)-1 {
        for j = i+1 to length(A) {
            gold = A[j] - A[i] if (gold > maxGoldSoFar) { maxGoldSoFar =
    gold }
    }}
     return maxGoldSoFar
```

What is the running time complexity of the procedure above? Write your answer as a Θ bound in terms of n.

The running time complexity is Θ(n^2). This is known because of the two for loops. One main for loop and one nested for loop. That means the inner for loop runs n times for each iteration of the outside for loop (0-n). n*n = n^2

(b) Explain (1–2 sentences) under what conditions on the contents of A the conningTheTownspeople algorithm will return 0 gold.

One of the conditions in which the algorithm will return 0, is if the array is all the same number. Another condition would be when elements are in descending order; i.e [12,10,0]

(c) Harry knows you know that conningTheTownspeople is wildly inefficient. After lecturing you for 20 minutes about the inefficiency of your algorithm, he suggests writing a function to make a new array M of size n such that

$$M[i] = \min_{0<=j<=i} A[j]$$

That is, M[i] gives the minimum value in the subarray of A[0..i]. What is the running time complexity of the pseudocode to create the array M? Write your answer as a Θ bound in terms of n.

The running time to create array M is *Θ(n) because it will only go through the array A once to append the minimum array values*

(e) Use the template code provided to determine and compare the runtimes for the functions in 2a and 2d. Explain your findings.

Comparing my runtime and the previous run times they don't appear to be too different for the lower values of n, its not until the size of the main array increases where part D is significantly faster.

3. (15 points) Consider the problem of linear search. The input is a sequence of n numbers A = (a1, a2, . . . , an) and a target value v. The output is an index i such that v = A[i] or the special value NIL if v does not appear in A.

(a) Write pseudocode for a simple linear search algorithm, which will scan through the input sequence A, looking for v.

```
def LinearSearch(A,v):
    for i in range(0,len(A)):
        if A[i]==v:
            return i
    return "NIL"
```

(b) Using a loop invariant, prove that your algorithm is correct. Be sure that your loop invariant and proof covers the initialization, maintenance, and termination conditions.

Claim: The function LinearSearch that takes a list A[0,...,n-1] and a value v, will return the index i in which A[i] = v. If the value v is not present in the array the value returns NIL.

Initialization:
Before this loop we assign i to the value of 0 to start at the beginning of the list A. The range will go until n which is the last index of the list A.

Maintenance: As the loop runs the function will check if the value of A[i] is equal to the requested value of v. As the loop iterates we know that the values A[0,...,i-1] do not contain the value v. The values A[i,...,n-1] are unknown. However, if A[i] == v, then the loop will break and the function will return the index of i.

Termination:
The loop terminates either when the value of v is found in the list, or when i = n-1.
i=n-1
A[0,...i]

It is then known that the list does not in fact contain the value of v, so the function returns “NIL”.

(20 points) Solve the following recurrence relations using any of the following methods: unrolling, tail recursion, recurrence tree (include tree diagram), or expansion. Each each case, show your work.

(a) T(n) = T(n − 2) + C n if n > 1, and T(n) = C otherwise

= T(n-3) + C(n-1) + Cn

= T(n-4) + C(n-2) + C(n-1) + Cn

.

.

.

= T(0) + C(0) + C(1) +C(2) + … C(n)
= C + C + 2C + … + nC
= C+(C * sum(K) for k =(1,...,n))
= C(n(n-1))/2) + C

(b) T(n) = 3T(n − 1) + 1 if n > 1, and T(1) = 3

= 3(3(T(n-2) + 1)) + 1
= 3^2 (T(n-2) + 1 + 3
= 3^3(T(n-3)) + 1 + 3 + 9

.

.

.

T(k) = 3^k (T(1)) + 3^k + 3^(k-1) … 3^2 + 3 + 1
T(k) = 3^k (3) + 3^k + 3^(k-1) … 3^2 + 3 + 1
T(n) = 3^n (3) + 3^n + 3^(n-1) … 3^2 + 3 + 1

Geometric recurrence ar^n-1
a=1 r=3

T(n) = 3* (3^n) + (1*3^(n)-1)/2

T(n) = 3* (3^n) + (3^n)/2 - ½

T(n) = (7(3^n) - 1) / 2

(c) T(n) = T(n − 1) + 2^n if n > 1, and T(1) = 2

= T(n-2) + 2^(n-1) + 2^(n)
= T(n-2) + 2^(n-1) + 2^(n)
= T(n-3) + 2^(n-2)+2^(n-1) + 2^(n)
.
.
.
T(k) = T(1) + 2^(k) + 2^(k-1) … 2^2 + 2^1
T(k) = 2 + 2^(k) + 2^(k-1) … 2^2 + 2^1
Geometric
a=1 r =2
a1=2

T(n) = 2((1-2^n)/(1-2))
T(n) = 2((1-2^n)/(-1))
T(n) = -2((1-2^n)

(d) T(n) = T(√ n) + 1 if n > 2 , and T(n) = 0 otherwise

= T(4√ n) + 1 + 1

= T(8√ n) + 1 + 1 + 1

T(r) = T(2^r√ n) + r

T(r) = T(2) + r

T(r) = 0 + r

T(n) = lg(n) - 1

5. (25 points) Asymptotic relations like O, Ω, and Θ represent relationships between functions, and these relationships are transitive. That is, if some f(n) = Ω(g(n)), and g(n) = Ω(h(n)), then it is also true that f(n) = Ω(h(n)). This means that we can sort functions by their asymptotic growth.1

Sort the following functions by order of asymptotic growth such that the final arrangement of functions g1, g2, . . . , g12 satisfies the ordering constraint g1 = Ω(g2), g2 = Ω(g3), . . . , g11 = Ω(g12).

e^n >= n^2 >= n! >=(3/2)^n >=lg(n!)>= n >= 2^(lgn) >= lg(n) >= (√ 2)^(lg(n)) >= n^(1/ lg(n))

n^2 >= n!
n! Eventually becomes greater than n^2 once n = 4

(3/2)^n eventually becomes greater than n^2 once n = 12

n >= 2^(lg(n)) are pairs