CSCI 3104 Spring 2018 Problem Set #2
Ali Ostlund (03/03)
Worked with: Kyle Ward, Hannah Haines

---

1. (20 pts total) 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) + Cn if n > 1, and T(n) = C otherwise

   $$T(n-2) = T(n-4) + C(n-2)$$
   $$T(n-4) = T(n-6) + C(n-4)$$
   $$T(n-4) = T(n-3) + C(n-2)$$
   $$\vdots$$
   $$T(n-2) = [T(n-6) + C(n-4)] + C(n-2)$$
   $$T(n) = [T(n-6) + C(n-4) + C(n-2)] + Cn$$
   $$T(n) = Cn + C(n-2) + C(n+4) + ... + 1$$
   $$\sum_{i=1}^{n} C(2i-1) = C\sum_{i=1}^{n} 2i - 1 = n^2 = T(n)$$
   $$= \Theta(n^2)$$

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

   $$T(n-1) = 9T(n-2) + 1 + 1$$
   $$T(n-2) = 3(3T(n-3) + 1) + 1$$
   $$T(n-3) = 3(3(3T(n-3) + 1) + 1) + 1$$
   $$= 9(3T(n-3) + 1) + 3 + 1)$$
   $$T(n) = 27T(n-3) + 9 + 3 + 1$$
   $$\sum_{i=0}^{n} C \cdot 3^i = C \cdot \sum_{i=0}^{n} 3^i = 3^n$$
   $$= \Theta(3^n)$$

   (c) T(n) = T(n 1) + 3^n if n > 1, and T(1) = 3

$T(n-1) = T(n-2) + 3^{n-1}$
$T(n-2) = T(n-3) + 3^{n-2}$
$T(n-3) = T(n-4) + 3^{n-3}$
$\vdots$
$T(n-2) = [T(n-4) + 3^{n-3}] + 3^{n-2}$
$T(n-1) = [T(n-4) + 3^{n-3} + 3^{n-2}] + 3^{n-1}$
$T(n) = [T(n-4) + 3^{n-3} + 3^{n-2} + 3^{n-1}] + 3^n$
$\sum_{i=1}^{n} C \cdot 3^i = C \cdot \sum_{i=1}^{n} 3^i = 3^n$
$= \Theta(3^n)$

(d) T(n) = T($n^{1/4}$) + 1 if n > 2 , and T(n) = 0 otherwise

$T(n-1) = T(n^{(\frac{1}{4})})^{\frac{1}{4}} + 1 + 1$
$T(n-2) = T((n^{\frac{1}{4}})^{(\frac{1}{4})})^{(\frac{1}{4})} + 1 + 1 + 1$
k = number of iterations
stop iterating when $n^{(\frac{1}{4k})} \leq 2$
$\log(n^{(\frac{1}{4k})}) \leq \log(2)$
$(\frac{1}{4k}) \cdot \log(n) \leq \log(2)$
$4^k \leq \frac{\log(n)}{\log(2)}$
$k \cdot \log(4) \leq \log(\log_2(n))$
$k \leq \frac{\log(\log_2(n))}{log(4)}$
$k \leq \log_4(\log_2(n))$
$T(n) = \Theta(\log_4(\log_2(n)))$

2. (10 pts) Consider the following function:

```
def foo(n):
if (n > 1)
print( hello )
foo(n/3)
foo(n/3)
```

In terms of the input n, determine how many times is hello printed. Write down a recurrence and solve using the Master method.

(a) Masters Method $T(n) = aT(\frac{n}{b}) + f(n)$
a = 2
b = 3
$T(n) = 2T(\frac{n}{3}) + F(n)$

F(n) = C(1) = 0

$= 2T(\frac{n}{3}) + C(1)$
$= 2T(\frac{n}{3}) + 0$
Rule: $C < \log_b(a) \Rightarrow \Theta(n^{\log_b(a)}$
Since $0 < \log_3(2) \Rightarrow T(n) = \Theta(n^{\log_3(2)})$
The algorithm will print "hello" $n^{\log_3(2)}$ times.

3. (30 pts) Professor McGonagall asks you to help her with some arrays that are raludominular. A raludominular array has the property that the subarray A[1..i] has the property that A[j] > A[j + 1] for 1 j ¡ i, and the subarray A[i..n] has the property that A[j] < A[j + 1] for i j ¡ n. Using her wand, McGonagall writes the following raludominular array on the board A = [7, 6, 4, 1, 2, 9, 5, 3, 10, 13], as an example.

(a) Write a recursive algorithm that takes asymptotically sub-linear time to find the minimum element of A.

```
Function(A, start, end):
  x = [(start + end) / 2]
  if A[x-1] < A[x] and A[x] < A[x+1]:
     Function(A start, x-1)
  if A[x-1] > A[x] and A[x] > A[x+1]:
     Function(A, x+1 ,end)
  if A[x-1] > A[x] and A[x] < A[x+1]:
     return x
```

(b) Prove that your algorithm is correct. (Hint: prove that your algorithms correctness follows from the correctness of another correct algorithm we already know.)

Proof by Induction:
BaseCase : n = 3 or the length of the sorted array is also three. Then x would have be the minimum value therefore would be the "spike" and the algorithm would return x.
InductionProcess : for values n $\geq$ 3, if A[x-1] < A[x] and A[x] < A[x+1] then the

first if statement is true and the minimum value is between A[left most side.... x-1] and it will recurse on the sub array A[begin.... x-1]. if A[x-1] > A[x] and A[x] > A[x+1] then the second if statement is true and the minimum value is between A[x+1.... right most side], and it will recurse on the sub array A[x+1.....end]. These two check and recursions will eventually be checking a subarray of size 3 because it will have to be between A[left side.... right side].

(c) Now consider the multi-raludominular generalization, in which the array contains k local minima, i.e., it contains k subarrays, each of which is itself a raludominular array. Let k = 2 and prove that your algorithm can fail on such an input.

[4,3,2,1,2,3,5,4,2,-1,2]
In this array, if the start value is 3, the algorithm will send you to the wrong minimum of 1 and not actually hit the real minimum of -1.

(d) Suppose that k = 2 and we can guarantee that neither local minimum is closer than n/3 positions to the middle of the array, and that the joining point of the two singly-raludominular subarrays lays in the middle third of the array. Now write an algorithm that returns the minimum element of A in sublinear time. Prove that your algorithm is correct, give a recurrence relation for its running time, and solve for its asymptotic behavior.

```
Func(A, start, end):
    x = [(start + end) / 2]
    if A[x-1] < A[x] and A[x] < A[x+1]:
       Function(A, start, x-1)
    if A[x-1] > A[x] and A[x] > A[x+1]:
       Function(A, x+1 ,end)
    if A[x-1] > A[x] and A[x] < A[x+1]:
       return x
    A = Function(A, start, x)
    B = Function(A,x, end)
    if( A > B):
       minimum of array = B
    else
       minimum of array = A
```

The function FUNC is a correct algorithm because FUNCTION from the question above is the correct implementation proven above.FUNC calls on FUNCTION once which is similar to FUCNCTIONS implementation, which is the exact same as the form of FUNCTION but FUNC is searching for the maximum value rather than the minimum. In each case you would just pass in different start and end values depending on which comparison the algorithm encounters. The terminating case for FUNCTION would be A[x-1] < A[x] > A[x+1], which is the same base case. This gets down to sub-arrays of 3 and implements opposite checks for each recursion pass.

Recurrence:

```
Func        T(N) = T(n/2) +1 using master method
            log_2(1) = 0 = Theta(n^log_2(1))

Function          log(n)
Function          log(n)
compare           O(1)
compare           O(1)
```

4. (15 pts extra credit) Asymptotic relations like O, , and represent relationships between functions, and these relationships are transitive. That is, if some f(n) = (g(n)), and g(n) = (h(n)), then it is also true that f(n) = (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 $= \Omega(g2), g2 = \Omega(g3), ..., g11 = \Omega(g12)$.
$\mid n! \mid n \mid n^{1.5} \mid 4^{\log *(n)} \mid 8^{\log(n)} \mid \log(n)! \mid (\frac{5}{4})^n \mid n^{\frac{1}{\log(n)}} \mid n\log(n) \mid \log(n!) \mid e^n \mid 42 \mid$
Give the final sorted list and identify which pair(s) functions f(n), g(n), if any, are in the same equivalence class, i.e., f(n) = Θ(g(n)).

(a) Sorted list :

$\mid n! \mid e^n \mid (\frac{5}{4})^n \mid \log(n)! \mid 8^{\log(n)} \mid n^{1.5} \mid \log(n!) \mid n\log(n) \mid n \mid 4^{\log *(n)} \mid n^{\frac{1}{\log(n)}} \mid 42 \mid$

Proof:

$$\lim_{n\to\infty}(\frac{e^n}{n!}) = 0 \Rightarrow n! = \Omega(e^n)$$

$$\lim_{n\to\infty}(\frac{(\frac{5}{4})^n}{e^n}) = 0 \Rightarrow e^n = \Omega((\frac{5}{4})^n)$$

$$\lim_{n\to\infty}(\frac{\log(n)!}{(\frac{5}{4})^n}) = 0 \Rightarrow (\frac{5}{4})^n = \Omega(\log(n)!)$$

$$\lim_{n\to\infty}(\frac{8^{\log(n)}}{\log(n)!}) = 0 \Rightarrow \log(n)! = \Omega(8^{\log(n)})$$

$$\lim_{n\to\infty}(\frac{n^{1.5}}{8^{\log(n)}}) = 0 \Rightarrow 8^{\log(n)} = \Omega(n^{1.5})$$

$$\lim_{n\to\infty}(\frac{\log(n!)}{n^{1.5}}) = 0 \Rightarrow n^{1.5} = \Omega(\log(n!))$$

$$\lim_{n\to\infty}(\frac{n\log(n)}{\log(n!)}) = 1 \Rightarrow \log(n!) = \Theta(n\log(n)) \text{ and } n\log(n) = \Theta(\log(n!))$$

$$\lim_{n\to\infty}(\frac{n}{n\log(n)}) = 0 \Rightarrow n\log(n) = \Omega(n)$$

$$\lim_{n\to\infty}(\frac{4^{\log*(n)}}{n}) = 0 \Rightarrow n = \Omega(4^{\log*(n)})$$

$$\lim_{n\to\infty}(\frac{n^{\frac{1}{\log(n)}}}{4^{\log*(n)}}) = 0 \Rightarrow 4^{\log*(n)} = \Omega(n^{\frac{1}{\log(n)}})$$

$$\lim_{n\to\infty}(\frac{42}{n^{\frac{1}{\log(n)}}}) = 42e \Rightarrow 42 = \Theta(n^{\frac{1}{\log(n)}}) \text{ and } n^{\frac{1}{\log(n)}} = \Theta(42)$$