CSCI 3104 Fall 2017 Homework 2
Austin Griffith (9/15)

---

1. *Solve the following recurrence relations using any of the following meth- ods: unrolling, tail recursion, recurrence tree (include tree diagram), or expansion. Each each case, show your work.*

   (a) *T(n)=T(n1)+2n if n¿1,and T(1)=2.*

```
T(n) = T(n-1) + 2^n
T(0) = 2^0 = 1
T(1) = 2^1 = 2
T(2) = T(1) + 2^2 = 2 + 2^2 = 6
T(3) = T(2) + 2^3 = 6 + 2^3 = 14
T(4) = T(3) + 2^4 = 14 + 2^4 = 30
...
T(n) = T(n-1) + 2^n = (2^n - 2) + 2^n
```

   (b) *T(n)=T*$(\sqrt{n}) + 1 if n > 2, and T(n) = 0 otherwise.$

T(n) = T$(\sqrt{n}) + 1 = T(n^1/2) + 1$

Unrolling step

= T(n$^1/2) + 1$

$= (T(n^1/4) + 1) + 1$

$= (T(n^1/8) + 1) + 2$

$= (T(n^1/16) + 1) + 3$

...

$= (T(n^(1/2^k)) + k$

```
 When n^1/(2^k) is <= 2, we have k >= log(log n)
therefore the run time would be :
```

T(n) $= \Theta(1) + log(logn) = \Theta(log(logn))$

2. *Consider the following function: def foo(n) if (n ¿ 1) print(hello) foo(n/4) foo(n/4) In terms of the input n, determine how many times is hello printed. Write down a recurrence and solve using the Master method.*

   Using the Master method where T(n) = aT(n/b) + f(n) we have:

   ```
   T(n) = 2T(n/4) + 1
   a = 2   b = 4   f(n) = 1
   ```

   $n^{\log_4 2}$ = $\mathrm{n}^{0.5}$

let $\epsilon = .5$

O($n^{\log_4 2-.5}$) = O($n^{\log_4 1.5}$)

therefore T(n) = Θ($n^{\log_4 1.5}$) by master method 1 O($n^{\log_4 1.5}$) grows faster than f(n). T(n) = O($n^{\log_4 1.5}$) hold true for case 1 of the Master method because n grow faster then f(n) which is 1.

3. *Professor McGonagall asks you to help her with some arrays that are spiked. A spiked 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 spiked array on the board A = [7, 8, 10, 15, 16, 23, 19, 17, 4, 1], as an example.*

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

```
findMax(A,low,high)
{
    mid = floor(low + high)/2
    if(A[m] > A[m+1] && A[m] > A[m-1])//found max
        return mid
    else if(A[m] > A[m+1] && A[m] < A[m-1])//go left
        return findMax(A,low,mid-1)
    else // go right
         return findMax(A,mid+1,high)
}
```

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

```
    Loop invariant = mid in array A[low...high]

Initialization:
low < mid < high in array A therefore max is in A[i...length(A)-1]

Maintenance:

Case 1: mid > mid + 1 and mid > mid -1 == target is
found continue to termination

Case 2: mid > mid + 1 and mid > mid - 1 == target must
be to the left of mid by the pre conditions of a
spiked array
```

```
Case 3: mid < mid + 1 and mid < mid - 1 == target must
be to the right of mid by the pre conditions of a
spiked array
```

Termination:
If the loop terminates, max was found because max is in array A by the initialization step and is either found, found to the right or found to the left of mid within array A[low...high]

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

```
Counter Example:

Let A = [1,2,9,10,8,11,7,6]

First Iteration:
mid = 3;    A[3] = 10
returns 10;
10 is not the max of array A
```

(d) *Suppose that $k = 2$ and we can guarantee that neither spike is closer than $n/4$ positions to the middle of the array, and that the joining point of the two singly-spiked subarrays lays in the middle half of the array. Now write an algorithm that returns the maximum 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.*

```
 findMax(A,low,high){

joinP = floor(low + high)/2
leftBound = floor(low + joinP)/2
rightBound = floor(high + joinP)/2
midLeft = floor(low + leftBound)/2
```

```
midRight = floor(high + rightBound)/2
maxLeft = 0
maxRight = 0

while(maxLeft =! 0 && maxRight =! 0){

    if(A[midLeft] > A[midLeft +1] && A[midLeft] > A[midLeft - 1]
        maxLeft = A[midLeft]

    else if(A[midLeft] > A[midLeft +1] && A[midLeft] < A[midLeft - 1]
        maxLeft = findMax(A,low,midLeft-1)

    else if(A[midLeft] < A[midLeft +1] && A[midLeft] > A[midLeft - 1]
        maxLeft = findMax(A,midLeft+1,high)

    else if(A[midRight] > A[midRight +1] && A[midRight] > A[midRight - 1]
        maxRight = A[midRight]

    else if(A[midRifht] > A[midRight +1] && A[midRight] < A[midRight - 1]
        maxRight = findMax(A,low,midRight-1)

    else if(A[midRight] < A[midRight +1] && A[midRight] > A[midRight - 1]
        maxRight = findMax(A,midRight+1,high)

    }

    if (maxLeft >= maxRight)
        return maxLeft

    else
        return maxRight
}
```

4. *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 arrange- ment of functions g1, g2, . . . , g12 satisfies the ordering constraint g1 = (g2), g2 = (g3), ..., g11 = (g12).*

   Some of the following identities were used to solve this problem:
   n$^{lg*lgn} = lg(n)^{lgn}$
   $n^2 = 4^{lgn}$
   $n = 2^{lgn}$
   $2^{\sqrt{2lgn}} = n^{\sqrt{2/lgn}}$
   $1 = n^{1/lgn}$

   $1 =$ n$^{1/logn} < (\sqrt{2})^{logn} < n = 2^{logn} < nlogn = log(n!) < n^2 < (logn)! < (3/2)^n < e^n < n!$