CPSC 320: Intermediate Algorithm Design and Analysis
Exercises with recurrence relations

1. $$T(n) = \begin{cases} 2T(\lfloor n/2 \rfloor) + 8T(\lfloor n/4 \rfloor) + n^2 \log n & \text{if } n \geq 4 \\ 1 & \text{if } n \leq 3 \end{cases}$$

2. $$T(n) = \begin{cases} T(n-1) + 6T(n-2) + 17 & \text{if } n \geq 3 \\ 7 & \text{if } n = 1 \\ 15 & \text{if } n = 2 \end{cases}$$

   Hint: prove that $T(n) \in O(3^n)$.

3. $$T(n) = \begin{cases} 2T(\lfloor 5n/9 \rfloor) + T(\lfloor 2n/9 \rfloor) + n^2 & \text{if } n \geq 9 \\ \Theta(1) & \text{if } n \leq 8 \end{cases}$$

4. Write recurrence relations describing the worst-case running time of the following algorithms in terms of n, where $n = \text{last} - \text{first} + 1$ (first and last will both be positions in an array). You can ignore floors and ceilings.

   a. This algorithm is really quite silly.

```
StrangeSum(A, first, last)

if first = last then
    return A[first]

n ⟵ last - first + 1
half ⟵ ⌊ n/2 ⌋
third ⟵ ⌊ n/3 ⌋

x ⟵ StrangeSum(A, first + third, last)
y ⟵ StrangeSum(A, first, first + half)
z ⟵ StrangeSum(A, first + half - third, first + half + third)
return x + y - z
```

   b. This one isn't any better.

```
Algorithm Bizarre(A, first, last)

if first = last then
    return A[first]

prod ⟵ 1
a ⟵ ⌊√(last - first + 1)⌋
for i ⟵ 1 to a do
```

```
    first ⟵ first + a/2
    last ⟵ last - a/2
    prod ⟵ prod * Bizarre(A, first, last)
return prod
```

c. Nor is this one

```
  Algorithm RudolfTheReindeer(A, start, n)
    x ⟵ 0
    for i ⟵ start to start + n do
        x ⟵ x + A[i]
    endfor

    t ⟵ n
    while (t > 0) do
        t ⟵ t/2
        x ⟵ x * RudolfTheReindeer(A, start + t, t)
    endwhile
```

d. Or this one

```
Algorithm ABitStrange(A,first,last)
    max ⟵ A[first]

    While (first < last) do
        mid ⟵ ⌊(first + last)/2⌋
        newMax ⟵ ABitStrange(A, first, mid)
        if (newMax > max) then
            max ⟵ newMax
        endif
        first ⟵ mid + 1
    Endwhile

    Return max
```

e. This one actually does something useful: it generates all of the sequences of n 0's and 1's that do not contain two consecutive 1's.

```
Algorithm GetAllPositionSequences(n)
    if (n == 0) then
        return ("")
    endif

    if (n == 1) then
```

```
    return ("0", "1")
endif

S ⟵ ∅

AllPositionSequences0 ⟵ GetAllPositionSequences(n-1)
for i ⟵ 0 to length[AllPositionSequences0]-1 do
    add "0" + AllPositionSequences0[i] to S
endfor

AllPositionSequences1 ⟵ GetAllPositionSequences(n-2)
for i ⟵ 0 to length[AllPositionSequences0]-1 do
    add "10" + AllPositionSequences1[i] to S
endfor

return S
```

After you have written the recurrence, use induction to prove that $T(n) \in O(\phi^n)$ where $\phi = (1 + \sqrt{5})/2$.

5. Determine whether or not each of the following recurrence relations can be solved by applying the Master theorem. Justify why or why not, and in the cases where the recurrence can be solved, give a Θ bound on the solution to the recurrence. Assume that in both cases, $T(n) \in \Theta(1)$ when n is sufficiently small.

   a. $T(n) = 4T(n/64) + n^{1/3} \log^3 n$

   b. $T(n) = 2T(\lfloor n/2 \rfloor) + n^2 \tau(n)$ where $\tau(n)$ is the number of 1's in the binary representation of n.

   c. $T(n) = 3T(\lfloor n/3 \rfloor) + n^2 \tau(n)$ where $\tau(n)$ is the number of 1's in the binary representation of n.