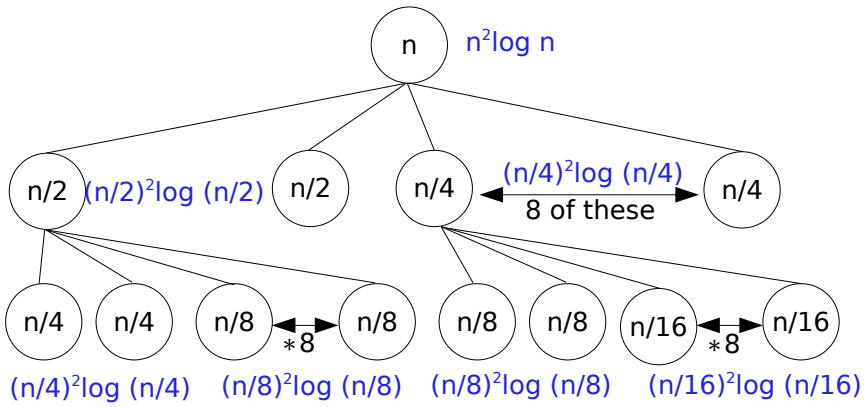
CPSC 320: Intermediate Algorithm Design and Analysis
Exercises with recurrence relations

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

Solution: We use a recursion tree. I have drawn the first three levels of the tree, skipping the middle 6 of the 8 repeated nodes on level 2, and only showing the children of two of these nodes on level 3.



The root does $n^2 \log n$ work. The sum of the work on level 2 is $2((n/2)^2 \log(n/2) + 8((n/4)^2 \log(n/4)) = \frac{n^2 \log(n/2)}{2} + \frac{n^2 \log(n/4)}{2}$ which is $\leq \frac{n^2 \log n}{2} + \frac{n^2 \log n}{2} = n^2 \log n$. The sum of the work on level 3 is similarly $\leq n^2 \log n$, etc.

The tree has $\log_2 n$ levels in total, which gives us an $O(n^2 \log^2 n)$ upper bound on $T(n)$. It turns out that this upper bound is tight (although the proof of the matching lower bound is ugly enough that I didn't want to ask you to do it).

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 $T(n) \in O(3^n)$.

Solution: We will show that $T(n) \leq c3^n - 3$ (note that the guess that most people would have started with, namely $T(n) \leq c3^n$, does not work as one ends up with an extra +17 at the end of the induction step, which can not be gotten rid of).

For the base case, we observe that $1 = T(1) \leq c3^1 - 3$ for every $c \geq 4$, and that $15 = T(2) \leq c3^2 - 3$ for every $c \geq 2$. Let us now do the induction step. Consider $k \geq 3$. Assume that $T(i) \leq c3^i - 3$ for every $i < k$, and look at $T(k)$. We have

$$\begin{aligned} T(k) &= T(k-1) + 6T(k-2) + 17 \\ &\leq (c3^{k-1} - 3) + 6(c3^{k-2} - 3) + 17 \\ &\leq c3^{k-1} - 3 + 6c3^{k-2} - 18 + 17 \\ &\leq c3^{k-1} + 2c3^{k-1} - 4 \\ &\leq 3c3^{k-1} - 4 \\ &\leq c3^k - 3 \end{aligned}$$

as required (for every value of c). Hence by the principle of mathematical induction $T(n) \leq c3^n - 3$ for every $c \geq 4$. Picking $c = 4$ satisfies both bases cases and the induction step, and so we conclude that $T(n) \leq 4 \cdot 3^n - 3$, and hence $T(n) \in O(3^n)$.

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

Solution: We use a recursion tree (you will need to imagine the picture). The cost at the root node is n^2. The root has 3 children: two of them will recurse on $5n/9$ elements, the third one of $2n/9$ elements. The sum of the costs on that level will thus be $2 \cdot (5n/9)^2 + (2n/9)^2$ which is (after simplification) $2n^2/3$.

Observing that the sum of the costs at the children of a node is thus $2/3$ the cost at the node itself, we can conclude the fact that the cost on the third level of the tree will be $(2/3)^2 n^2$, the cost on the fourth level will be $(2/3)^3 n^2$, etc. This gives us a geometric series, whose sum is

$$\leq n^2/(1 - 2/3) = 3n^2.$$

Therefore $T(n) \in O(n^2)$. Moreover, $T(n) \geq n^2$ (the cost at the root) and so $T(n) \in \Omega(n^2)$. This shows that $T(n) \in \Theta(n^2)$.

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
```

Solution: Omitting the floors and ceilings for simplicity, the recurrence relation is

$$T(n) = \begin{cases} 2T(2n/3) + T(n/2) + \Theta(1) & \text{if } n \geq 2 \\ \Theta(1) & \text{if } n = 1 \end{cases}$$

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
```

Solution: Omitting the floors and ceilings for simplicity, the solution is

$$T(n) = \begin{cases} \Theta(\sqrt{n}) + \sum_{i=1}^{\sqrt{n}} T(n - i\sqrt{n}) & \text{if } n \geq 2 \\ \Theta(1) & \text{if } n \leq 1 \end{cases}$$

which can be simplified slightly to

$$T(n) = \begin{cases} \sum_{i=0}^{\sqrt{n}-1} T(i\sqrt{n}) & \text{if } n \geq 2 \\ \Theta(1) & \text{if } n \leq 1 \end{cases}$$

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
```

Solution: $$T(n) = \begin{cases} \Theta(1) & \text{if } n = 0 \\ \Theta(n) + \sum_{j=1}^{1+\lfloor \log_2 n \rfloor} T(n/2^j) & \text{if } n \geq 1 \end{cases}$$

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
```

Solution: The algorithm first recurses on $\lceil n/2 \rceil$ elements, then on $\lceil \lfloor n/2 \rfloor /2 \rceil$ elements, then on $\lceil \lfloor n/4 \rfloor /2 \rceil$, etc. We thus get the recurrence

$$T(n) = \begin{cases} \sum_{i=0}^{\lfloor \log_2 n \rfloor} \left(\Theta(1) + T(\lceil \lfloor n/2^i \rfloor /2 \rceil)\right) & \text{if } n \geq 2 \\ \Theta(1) & \text{if } n = 1 \end{cases}$$

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 $\phi = (1 + \sqrt{5})/2$.

Solution: The recurrence is

$$T(n) = \begin{cases} T(n-1) + T(n-2) & \text{if } n \geq 2 \\ 2 & \text{if } n = 1 \\ 1 & \text{if } n = 0 \end{cases}$$

We use the strong form of mathematical induction to prove that $T(n) \leq c\phi^n$ for some positive real number c. We start by the induction step. Consider an unspecified $n \geq 2$. Then

$$\begin{aligned} T(n) &= T(n-1) + T(n-2) \\ &\leq c\phi^{n-1} + c\phi^{n-2} \\ &= c\phi^{n-2}(\phi + 1) \end{aligned}$$

$$= c\phi^{n-2}\left(\frac{1+\sqrt{5}}{2}+1\right)$$
$$= c\phi^{n-2}\left(\frac{3+\sqrt{5}}{2}\right)$$

Now, observe that

$$\begin{aligned}\phi^2 &= \frac{(1+\sqrt{5})^2}{4}\\ &= \frac{1+5+2\sqrt{5}}{4}\\ &= \frac{6+2\sqrt{5}}{4}\\ &= \frac{3+\sqrt{5}}{2}\end{aligned}$$

and therefore

$$\begin{aligned}T(n) &\leq c\phi^{n-2}\phi^2\\ &= c\phi^n\end{aligned}$$

as required for the induction step (this works for every $c > 0$). We have two base cases to verify. For $n = 1$, we want

$$T(1) = 2 \leq c\phi$$

which is true as long as $c \geq 2/\phi$. Choose $c = 2/\phi$ also works for $n = 0$ since

$$T(0) = 1 \leq 2/\phi \times \phi^0 = 2/\phi \approx 1.236$$

This completes the proof by mathematical induction.

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

      Solution: Since $n^{1/3} \in \Theta(n^{\log_{64} 4})$, we are in case 2 of the Master theorem, and so $T(n) \in \Theta(n^{1/3}\log^4 n)$.

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.

Solution: We will show that Case 3 of the Master theorem can be applied to this recurrence. We have $a = 2$, $b = 2$ and $f(n) = n^2\tau(n)$. Clearly $f(n) \geq n^2$, and hence $f(n) \in \Omega(n^{1+\varepsilon})$ for $\varepsilon = 0.5$. Now we check the regularity condition. The binary representation of $\lfloor n/2 \rfloor$ is the same as that of n, except that the last bit is missing. Hence $\tau(\lfloor n/2 \rfloor) \leq \tau(n)$, which means that

$$\begin{aligned} af(\lfloor n/b \rfloor) &= 2f(\lfloor n/2 \rfloor) \\ &= 2\lfloor n/2 \rfloor^2 \tau(\lfloor n/2 \rfloor) \\ &\leq 2(n/2)^2\tau(n) \\ &= \frac{1}{2}n^2\tau(n) \end{aligned}$$

and so $af(n/b) < 0.8f(n)$. Hence the regularity condition holds, which means that $T(n) \in \Theta(n^2\tau(n))$.

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.

Solution: We have $a = 3$, $b = 3$, and $n^{\log_b a} = n^{\log_3 3} = n$. Now, $f(n) = n^2\tau(n) \geq n^2$. Hence $f(n) \in \Omega(n^{\log_b a + \varepsilon})$ for $\varepsilon = 1$. This means that the only possible case of the Master theorem that might apply is case 3.

Let us now verify the regularity condition. Consider a value of n that is a power of 2 ($n = 2^t$). Clearly $\tau(n) = 1$. What is $\tau(\lfloor n/3 \rfloor)$? Observing that 1/3 in binary is $0.0101010101010101\ldots$, we can see that $\tau(n/3) \approx t/2$. So

$$af(\lfloor n/b \rfloor) = 3f(\lfloor n/3 \rfloor) \approx 3(n/3)^2 t/2 = (n^2/3)\log n.$$

Clearly this is not less than $\delta f(n) = \delta n^2$ for any constant $\delta < 1$. Hence the regularity condition does not hold, which means that we can not apply case 3 of the Master theorem. Since this was the only case that might have applied, we can not apply the Master theorem.