CSE 332 - Section 3 Worksheet

0. Recurrence Relations

a) Find a recurrence T(n) modeling the worst-case runtime complexity of f(n)

```
f(n) {
  if (n <= 0) {
    return 1
  }
  return 2 * f(n - 1) + 1
}
```

$$T(n) = \begin{cases} c_0 & \text{if } n \leq 0 \\ T(n-1) + c_1 & \text{otherwise} \end{cases}$$

b) Find a recurrence T(n) modeling the worst-case runtime complexity of g(n)

```
g(n) {
  if (n <= 1) {
    return 1000
  }
  if (g(n/3) > 5) {
    for (int i = 0; i < n; i++) {
           println(“Yay”)
    }
    return 5 * g(n/3)
  } else {
    for (int i = 0; i < n * n; i++) {
           println(“Yay)
    }
    return 4 * g(n/3)
  }
```

$$T(n) = \begin{cases} c_0 & \text{if } n \leq 1 \\ 2T(\frac{n}{3}) + c_1 n + c_2 & \text{otherwise} \end{cases}$$

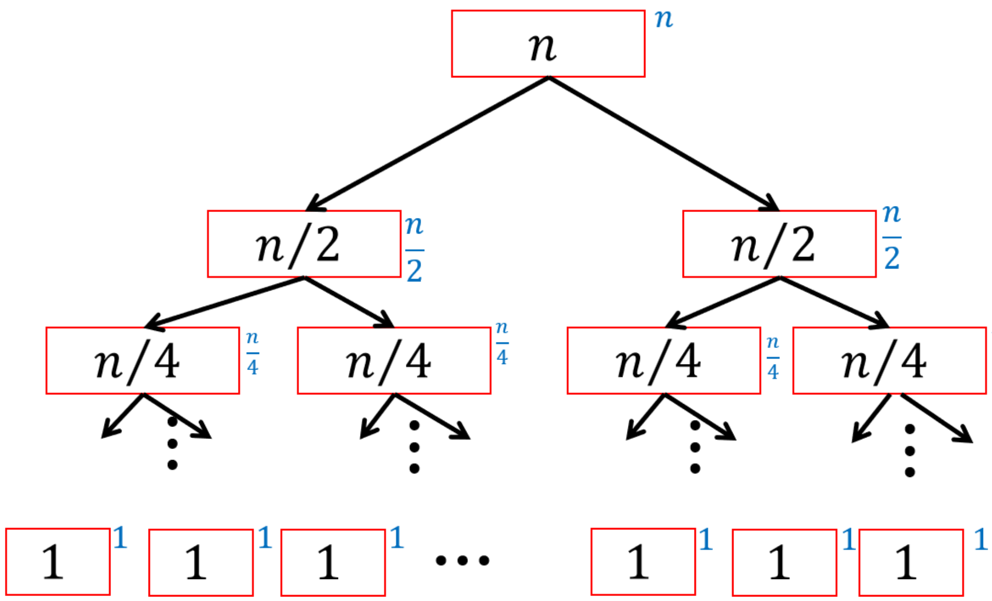
CSE 332 - Section 3 Worksheet

1. Tree Method

For each of the following recurrence relations, use the tree method to convert it to closed form:

a)

$$T(n) = \begin{cases} 1 & \text{if } n \leq 1 \\ 2T\left(\frac{n}{2}\right) + n & otherwise \end{cases}$$



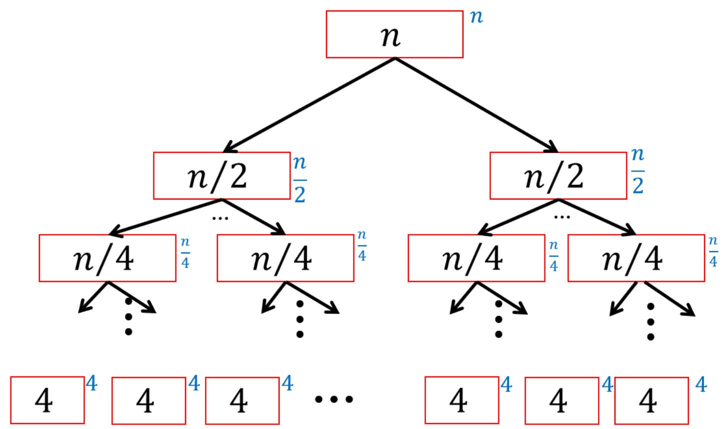
Summing up each row we get n work per level.

The height of the tree will be $\log_2 n$

Summing up all of the rows gives a running time of $\Theta(n \log n)$

b)

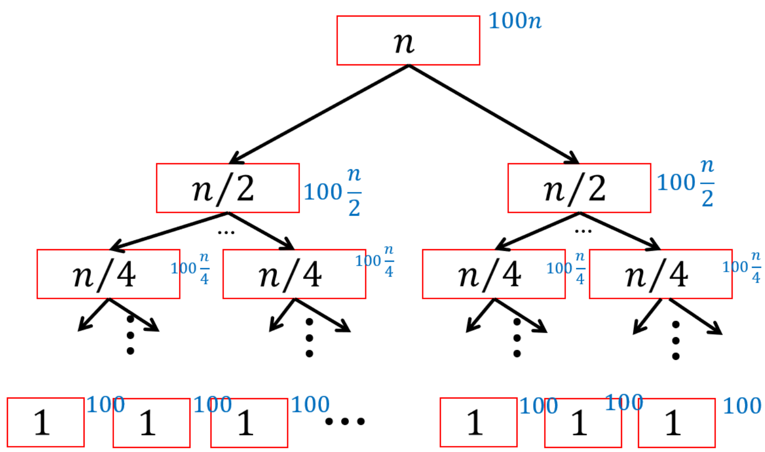
$$T(n) = \begin{cases} 4 & \text{if } n \le 4 \\ 2T\left(\frac{n}{2}\right) + n & otherwise \end{cases}$$



The only difference compared to the previous tree is that we have $\log_2 n - 2$ levels in this tree. With 2 fewer levels, the running time will be $n \log_2 n - 2n$ which is still $\Theta(n \log n)$

c)

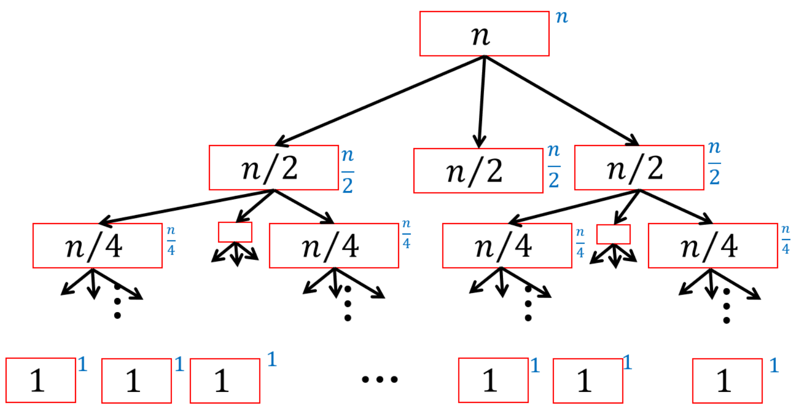
$$T(n) = \begin{cases} 100 & \text{if } n \le 1 \\ 2T\left(\frac{n}{2}\right) + 100n & otherwise \end{cases}$$



The only difference compared to the tree in problem a is that we do $100n$ work per level. This means that our running time will be $100\, n \log_2 n$ which is still $\Theta(n \log n)$

CSE 332 - Section 3 Worksheet

d)
$$T(n) = \begin{cases} 1 & \text{if } n \leq 1 \\ 3T\left(\frac{n}{2}\right) + n & otherwise \end{cases}$$



At each level the number of nodes in the tree will be triple the number of nodes at the previous level. If we consider the root to be at level 0 then the number of nodes on level i is 3^i. Additionally, the size of each subproblem will be half that of its parent, and so the size of each node on level i is $\frac{n}{2^i}$, which means there is $\frac{n}{2^i}$ non-recursive work.

The total work done on level i is therefore $\left(\frac{3}{2}\right)^i n$. Because there are $\log_2 n$ levels the solution is given by the sum

$$n \sum_{i=0}^{\log_2 n - 1} \left(\frac{3}{2}\right)^i$$

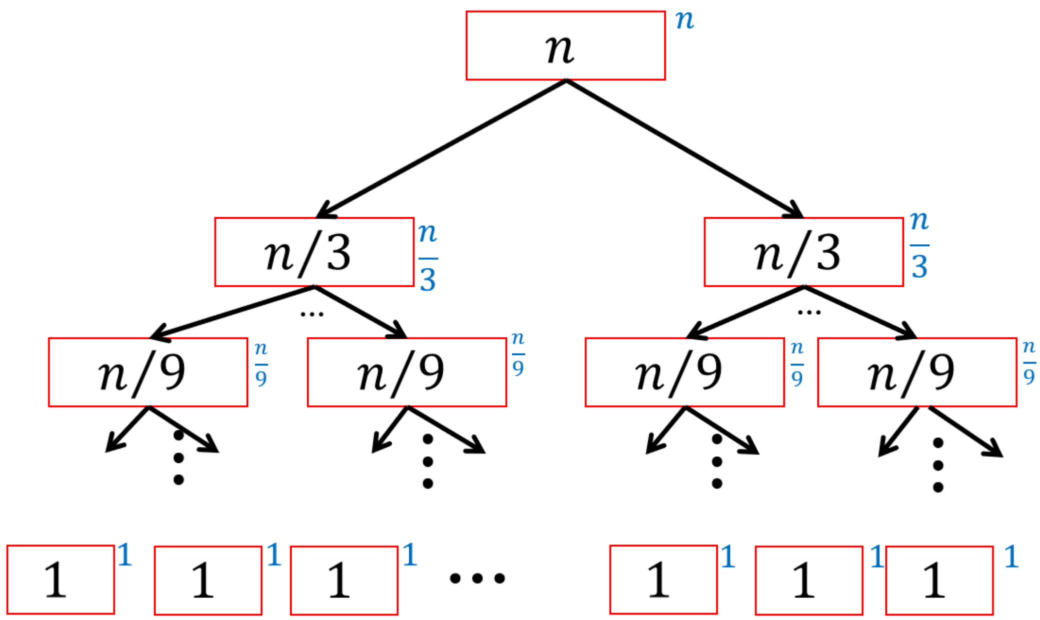
Applying the geometric series formula we get

$$n\left(\frac{1-\left(\frac{3}{2}\right)^{\log_2 n}}{1-\frac{3}{2}}\right) = 2n\left(\frac{3^{\log_2 n}}{2^{\log_2 n}} - 1\right) = 2n\left(\frac{n^{\log_2 3}}{n} - 1\right) = 2n^{\log_2 3} - 2n$$

This means that the solution is $\Theta\left(n^{\log_2 3}\right)$.

CSE 332 - Section 3 Worksheet

e)
$$T(n) = \begin{cases} 1 & \text{if } n \le 1 \\ 2T\left(\frac{n}{3}\right) + n & otherwise \end{cases}$$



At each level the number of nodes in the tree will be double the number of nodes at the previous level. If we consider the root to be at level 0, then the number of nodes on level i is 2^i. Additionally, the size of each subproblem will be one third that of its parent, and so the size of each node on level i is $\frac{n}{3^i}$, which means there is $\frac{n}{3^i}$ non-recursive work.

The total work done on level i is therefore $\left(\frac{2}{3}\right)^i n$.

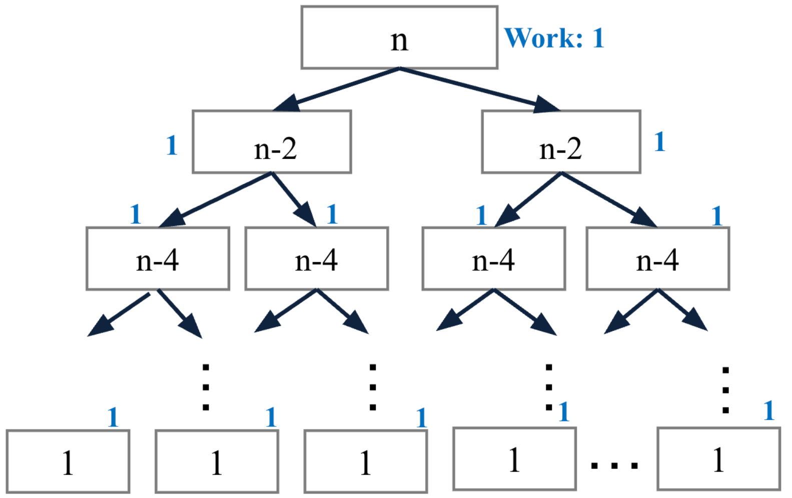
Because there are $\log n$ levels the solution is given by the sum

$$\sum_{i=0}^{\log_3 n - 1} \left(\frac{2}{3}\right)^i n = n \sum_{i=0}^{\log_3 n - 1} \left(\frac{2}{3}\right)^i = n * c$$

Observe that this is a geometric series with a ratio less than 1, and so the sum is upper-bounded by a constant.

This means that the solution is $\Theta(n)$.

f)
$$T(n) = \begin{cases} 1 & n \le 1 \\ 2T(n-2) + 1 & otherwise \end{cases}$$



Each node will have two child nodes since there are two recursive calls from the recurrence relation. At each level, each node will have work of 1, so the total work per level is 2^i. The input size decreases by two for every recursive call and reaches the base case when the input is 1, so the total number of calls will be around n/2. Combining these information, we can conclude that T(n) = $\sum_{i=0}^{n/2-1} 2^i$. Using the geometric serie sum, we can further simplify T(n) to $\sqrt{2}^n - 1$, which is in $\Theta(\sqrt{2}^n)$.

2. Putting It All Together

Consider the function $f(n)$. Find a recurrence modeling the worst-case runtime of this function and then find a Big-Oh bound for this recurrence.

```
f(n) {
    if (n <= 1) {
        return 0
    }
    int result = f(n/2)
    for (int i = 0; i < n; i++) {
        result *= 4
    }
    return result + f(n/2)
}
```

a) Find a recurrence $T(n)$ modeling the *worst-case runtime complexity* of $f(n)$

CSE 332 - Section 3 Worksheet

We look at the three separate components (base case, non-recursive work, recursive work). The base case is a constant amount of work, because we only do a return statement. We'll label it c_0. The non-recursive work is a constant amount of work (we'll call it c_1) for the assignments and `if` tests and a constant (we'll call c_2) multiple of n for the loops. The recursive work is $2T\left(\frac{n}{2}\right)$.

Putting these together, we get:

$T(n) = c_0$, if 1

$T(n) = 2T\left(\frac{n}{2}\right) + c_2 n + c_1$, otherwise

b) Use your answer in part (a) to find a closed form for $T(n)$

$$
\begin{aligned}
T(n) &= \sum_{i=0}^{\log_2(n)-1} 2^i \left(c_2\left(\frac{n}{2^i}\right) + c_1\right) \\
&= \sum_{i=0}^{\log_2(n)-1} \left(c_2 n + 2^i \cdot c_1\right) \\
&= c_2 n \log_2(n) + c_1 \sum_{i=0}^{\log_2(n)-1} 2^i \\
&= c_2 n \log_2(n) + c_1 \frac{1 - 2^{\log_2(n)}}{1-2} \\
&= c_2 n \log_2(n) + c_1 (n-1) \\
&\in \Theta(n \log n)
\end{aligned}
$$