CSE 332 - Section 2 Worksheet

Part One: Algorithm Analysis

0. True/False Questions

Suppose that $f(n) \in O(n)$. Indicate if each statement is guaranteed to be true, guaranteed to be false, or might be either.

a) $f(n) \in O(n^2)$

Always True | Sometimes True | Always False

b) $f(n) \in O(4n)$

Always True | Sometimes True | Always False

c) $f(n) \in O(\log n)$

Always True | Sometimes True | Always False

d) $f(n) \in O(n \log n)$

Always True | Sometimes True | Always False

e) $f(n) \in O(1)$

Always True | Sometimes True | Always False

f) $f(n) \in \Omega(n)$

Always True | Sometimes True | Always False

g) $f(n) \in \Omega(\log n)$

Always True | Sometimes True | Always False

h) $f(n) \in \Omega(n^2)$

Always True | Sometimes True | Always False

CSE 332 - Section 2 Worksheet

1. Big-Oh Proofs

For each of the following, prove that $f(n) \in \mathcal{O}(g)$:

a) $f(n) = 7n$ $\quad\quad$ $g(n) = \frac{n}{10}$

Let $c = 70$ and $n_0 = 1$

Next we show that $\forall n > n_0$ we have that $7n \leq c \cdot \frac{n}{10}$

We start with the right-hand side of the inequality.

$$70 \cdot \frac{n}{10} = 7n$$

Since $7n = c \cdot \frac{n}{10}$, then certainly $7n \leq c \cdot \frac{n}{10}$

And so $7n \in O\left(\frac{n}{10}\right)$

b) $f(n) = 1000$ $\quad\quad$ $g(n) = 3n^2$

Let $c = 1$ and $n_0 = 20$

Next we show that $\forall n > n_0$ we have that $1000 \leq c \cdot 3n^2$

We start with the right-hand side of the inequality.

$$c \cdot 3n^2 = 3n^2$$

When $n = 20$ we have $3n^2 = 3 \cdot 400 = 1200$, and when $n > 20$ we have that $3n^2 > 1200$

And so $1000 \in O(3n^2)$

c) $f(n) = 2^n$ $\quad\quad$ $g(n) = 3^{2n}$

Let c $= 1$ and $n_0 = 2$

Next, we show that $\forall_n > n_0$ we have that $2^n \leq c * 3^{2n}$

We start with the right-hand side of the inequality.

$2^n \leq 1 * 3^{2n}$

$2^n \leq 9^n$

This is certainly true $\forall_n >= 2$

d) $f(n) = 7n^2 + 3n$ $\quad\quad$ $g(n) = n^4$

Let $c = 10$ and $n_0 = 1$
Next we show that $\forall n > n_0$ we have that $7n^2 + 3n \leq c \cdot n^4$
We start with the left-hand side of the inequality.
$7n^2 + 3n \leq 7n^2 + 3n^2$ because $n^2 > n$ whenever $n > 1$.
Likewise $10n^2 \leq 10n^4$ whenever $n > 1$
And so $7n^2 + 3n \in O(n^4)$

e) $f(n) = n + 2n \lg n$ $g(n) = n \lg n$

Let $c = 3$ and $n_0 = 2$
Next we show that $\forall n > n_0$ we have that $n + 2n \log_2 n \leq c \cdot n \log_2 n$
We start with the left-hand side of the inequality.
$n + 2n \log_2 n \leq 3n \log_2 n$ because $\log_2 n > 1$ whenever $n > 2$
And so $n + 2n \log_2 n \in O(n \log_2 n)$

2. Big-Theta Proofs

For each of the following, prove that $f(n) \in \Theta(g)$:

a) $f(n) = 7n$ $g(n) = \frac{n}{10}$

The O portion was done in 1a, so all that remains is to show $7n \in \Omega\left(\frac{n}{10}\right)$
Again, let $c = 70$ and $n_0 = 1$
Next we show that $\forall n > n_0$ we have that $7n \geq c \cdot \frac{n}{10}$
We start with the right-hand side of the inequality.

$$70 \cdot \frac{n}{10} = 7n$$

Since $7n = c \cdot \frac{n}{10}$, then certainly $7n \geq c \cdot \frac{n}{10}$
And so $7n \in \Omega\left(\frac{n}{10}\right)$

b) $f(n) = n^3 + 10n$ $g(n) = 3n^3$

CSE 332 - Section 2 Worksheet

First we will show that $n^3 + 10$ belongs to $O(3n^3)$
Let $c = 1$ and $n_0 = 3$
Next we show that $\forall n > n_0$ we have that $n^3 + 10 \leq c \cdot 3n^3$
We start with the left-hand side of the inequality.
$n^3 + 10 \leq 2n^3$ because $n^3 > 10$ whenever $n \geq 3$. Since $2n^3 \leq 3n^3$ we can conclude $n^3 + 10 \in O(3n^3)$.

Next we will show that $n^3 + 10$ belongs to $\Omega(3n^3)$
Let $c = \frac{1}{3}$ and $n_0 = 1$
Next we show that $\forall n > n_0$ we have that $n^3 + 10 \geq c \cdot 3n^3$
We start with the left-hand side of the inequality.
Observe that $n^3 + 10 \geq \frac{1}{3} \cdot 3n^3$ for all positive values of n, therefore $n^3 + 10 \in \Omega(3n^3)$.

CSE 332 - Section 2 Worksheet

3. Algorithm Running Time

Consider the following method which finds the number of unique Strings within a given array of length n.

```
1  int numUnique(String[] values) {
2      boolean[] visited = new boolean[values.length]
3      for (int i = 0; i < values.length; i++) {
4          visited[i] = false
5      }
6      int out = 0
7      for (int i = 0; i < values.length; i++) {
8          if (!visited[i]) {
9              out += 1
10             for (int j = i; j < values.length; j++) {
11                 if (values[i].equals(values[j])) {
12                     visited[j] = true
13                 }
14             }
15         }
16     }
17     return out;
18 }
```

Determine a $\Theta(\cdot)$ bound on each function below. Start by (1) constructing an equation that models each function then (2) simplifying and finding a closed form.

a) $f(n)$ = the worst-case runtime of `numUnique`

The worst case occurs when all strings in the array are unique. The running time will be quadratic because:

- The for-loop beginning on line 7 runs n times regardless of the inputs
- Visited[i] becomes true on line 12 whenever the string at index i matches some previous string. By having all strings be unique no indices of visited will become true, and so we will always enter the if statement on line 8
- The for loop on line 10 will occur n-i times, which is asymptotically the same as n (certainly not more than n and certainly also more than n/2 for at least half of the iterations)
- Between the loop on lines 7 and 10, we have quadratic running time.

b) $g(n)$ = the best-case runtime of `numUnique`

The best case occurs when all strings match. The running time will be linear because:

- In the first iteration of the for-loop on line 7 we will mark visited[j] true for every index in the visited array, this means that the loop on line 10 will only ever occur once.

CSE 332 - Section 2 Worksheet

Part Two: Recurrences

0. Recurrence Relations

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

```
1  f(n) {
2    if (n <= 0) {
3      return 1
4    }
5    return 2 * f(n - 1) + 1
6  }
```

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

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

```
1  g(n) {
2    if (n <= 10000) {
3      return 1000
4    }
5    if (g(n/3) > 5) {
6      for (int i = 0; i < n; i++) {
7             println("Yay")
8      }
9      return 5 * g(n/3)
10   } else {
11     for (int i = 0; i < n * n; i++) {
12            println("Yay)
13     }
14     return 4 * g(n/3)
15   }
```

CSE 332 - Section 2 Worksheet

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

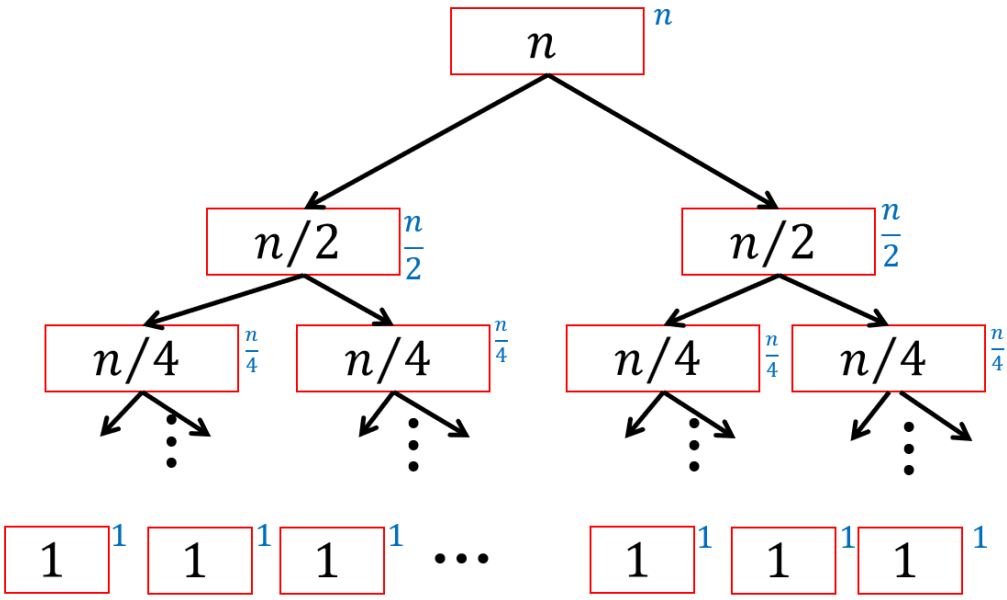
`if n <= 10000 not 1 here`

c) Tree Method

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

f)

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



Solving the Summation

$$T(n) = \sum_{i=0}^{\log(n)-1} n$$

$$= [(log_2(n) - 1) + 1] \cdot n$$

$$= n \cdot log_2(n)$$

$$\in \Theta(nlogn)$$

CSE 332 - Section 2 Worksheet

Summing up each row we get n work per level.

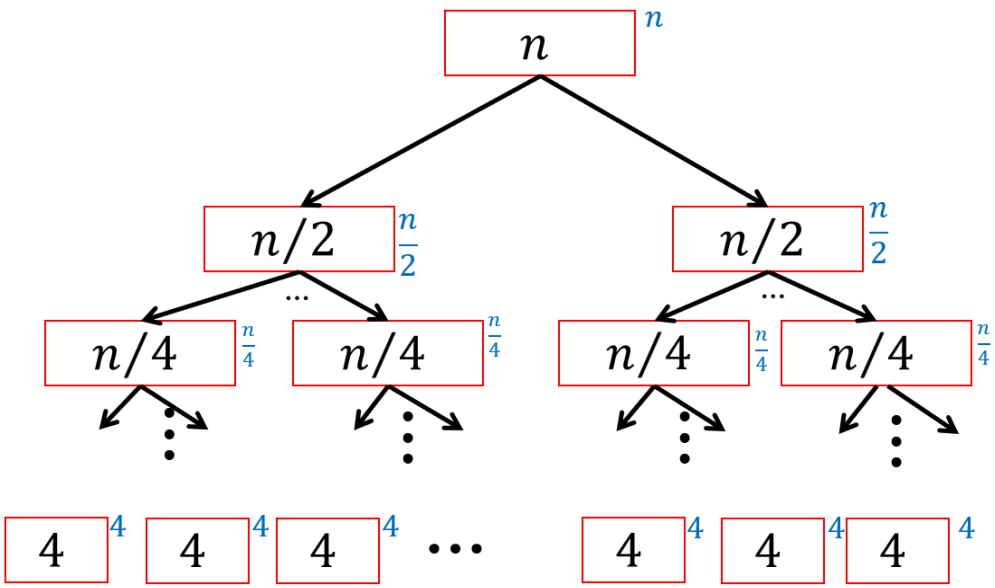
The height of the tree will be $\log_2 n$

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

CSE 332 - Section 2 Worksheet

g)

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



Solving the Summation

$$T(n) = \sum_{i=0}^{\log_2(n)-3} n$$

$$= [(log_2(n) - 3) + 1] \cdot n$$

$$= n \cdot log_2(n) - 2n$$

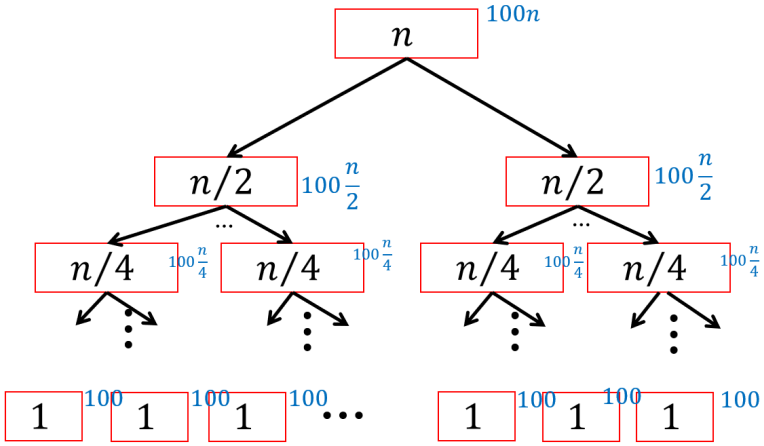
$$\in \Theta(nlogn)$$

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 $\Theta(n \log n)$

CSE 332 - Section 2 Worksheet

h)

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



Solving the Summation

$$T(n) = \sum_{i=0}^{\log_2(n)-1} 100n$$

$$= [(log_2(n) - 1) + 1] \cdot 100n$$

$$= 100n \cdot log_2(n)$$

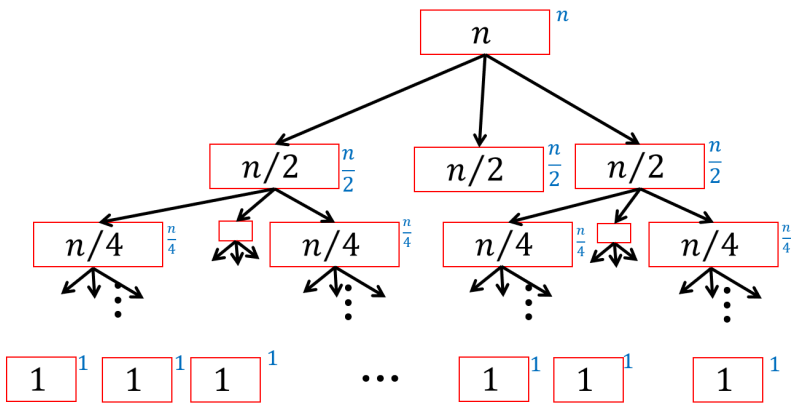
$$\in \Theta(nlogn)$$

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 2 Worksheet

i)

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



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

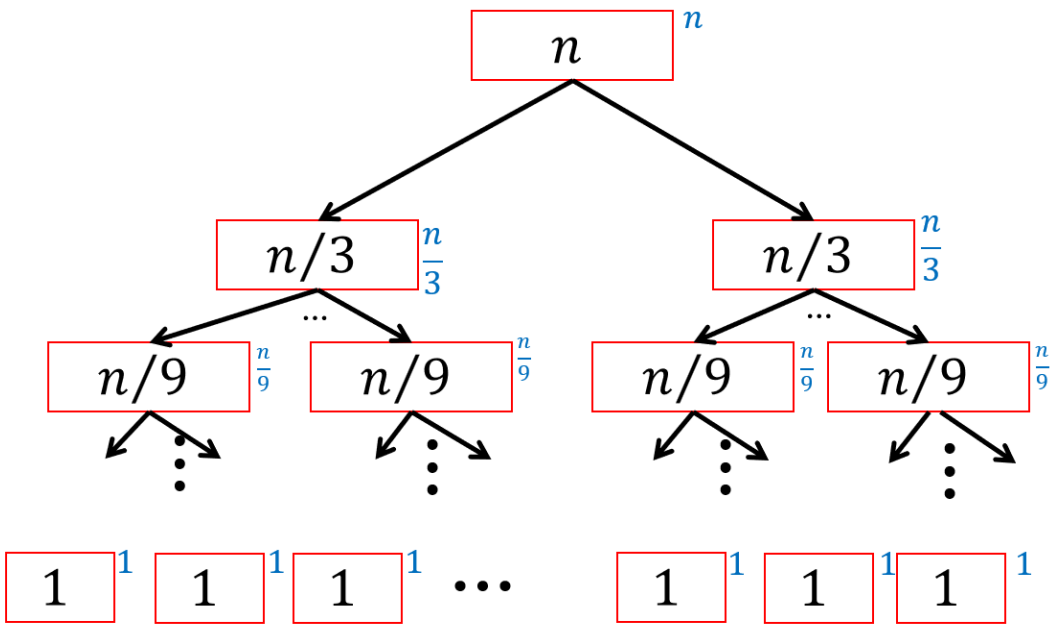
$$n\left(\frac{1-\left(\frac{3}{2}\right)^{\log_2 n}}{1-\frac{3}{2}}\right) = 2n\left(\frac{3^{\log_2 n}}{2^{\log_2 n}} - 1\right) = 2n\left(\frac{n^{\log_2 3}}{n} - 1\right) = 2n^{\log_2 3} - 2n$$

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

CSE 332 - Section 2 Worksheet

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

j)



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

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

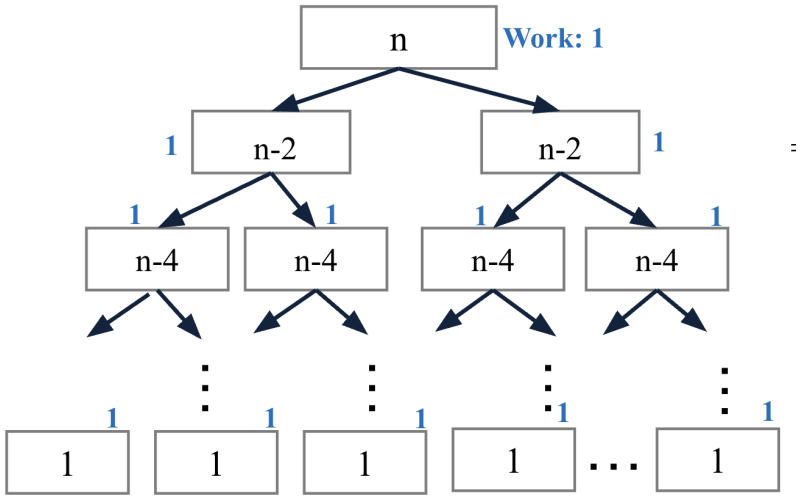
Observe that this is a geometric series with a ratio less than 1, and so the sum is upper-bounded by a constant.

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

CSE 332 - Section 2 Worksheet

k)

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



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

```
1  f(L, start, j) { // n = j - start
2      if (j - start <= 1) {
3          return 0
4      }
5      int result = f(L, start, j/2)
6      for (int i = start; i < j; i++) {
7          result *= 4
8      }
9      return result + f(L, start, j/2)
10 }
```

CSE 332 - Section 2 Worksheet

- Find a recurrence $T(n)$ modeling the *worst-case runtime complexity* of $f(L, j)$ where $n = j - start$

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

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

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