CSE 332: Data Structures and Parallelism

Section 3: Recurrences and Closed Forms

Terminology	Recurrence Function/Relation	General formula	Closed form
Definition	Piecewise function that mathematically models the runtime of a recursive algorithm (might want to define constants)	Function written as the number of expansion i and recurrence function (might have a summation)	General formula evaluated without recurrence function or summations (force them to be in terms of constants or n)
Example	$T(n) = c_1$, for $n = 1$ $T(n) = T\left(\frac{n}{2}\right) + c_2$, otherwise	$T(n) = T\left(\frac{n}{2^i}\right) + i \cdot c_2$	Let $i = \log_2 n$, $T(n) = T\left(\frac{n}{2^{\log_2 n}}\right) + \log_2 n \cdot c_2$ $= T(1) + \log_2 n \cdot c_2$ $= c_1 + \log_2 n \cdot c_2$

0. Not to Tree

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

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

b) Find a closed form for $T(n)$

1. To Tree

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

```
h(n) {
    if (n <= 1) {
        return 1
    } else {
        return h(n/2) + n + 2*h(n/2)
    }
}
```

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

b) Find a closed form for $T(n)$

2. To Tree or Not to Tree

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 $f(n)$

b) Find a closed form for $T(n)$

3. Big-Oof Bounds

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 = 0
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < i; j++) {
            result += j
        }
    }
    return f(n/2) + result + f(n/2)
}
```

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

b) Find a Big-Oh bound for your recurrence.

4. Odds Not in Your Favor

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

```
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)
    }
}
```

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

b) Find a closed form for $T(n)$