

Section 3: BSTs and Recurrences Solutions

0. Interview Question: Binary Search Trees

Write pseudo-code to perform an in-order traversal in a binary search tree without using recursion.

Solution:

This algorithm is implemented as the BST Iterator in P2. Check it out!

1. Recurrences and Closed Forms

For the following code snippet, find a recurrence for the worst case runtime of the function, and then find a closed form for the recurrence.

Consider the function f :

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

- Find a recurrence for $f(n)$.

Solution:

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

- Find a closed form for $f(n)$.

Solution:

Unrolling the recurrence, we get $T(n) = \underbrace{c_1 + c_1 + \dots + c_1}_{n \text{ times}} + c_0 = c_1 n + c_0$.

2. Recurrences and Big-Oh Bounds

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

```
1 f(n) {
2   if (n == 0) {
3     return 0
4   }
5
6   int result = 0
7   for (int i = 0; i < n; i++) {
8     for (int j = 0; j < i; j++) {
9       result += j
10
11     }
12   }
13   return f(n/2) + result + f(n/2)
14 }
```

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

Solution:

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 $\sum_{i=0}^{n-1} i = \frac{n(n-1)}{2}$ for the loops. The recursive work is $2T(\frac{n}{2})$.

Putting these together, we get:

$$T(n) = \begin{cases} c_0 & \text{if } n = 0 \\ 2T(\frac{n}{2}) + c_2 \frac{n(n-1)}{2} + c_1 & \text{otherwise} \end{cases}$$

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

Solution:

Since we only want a Big-Oh, we can actually leave off lower-order terms when doing our analysis, as they won't affect the runtime bounds; so, we can ignore the constants c_1 and c_2 in our analysis.

Note that $\frac{n(n-1)}{2} = \frac{n^2}{2} - \frac{n}{2} \in \mathcal{O}(n^2)$. We can, again, ignore the lower-order term ($\frac{n}{2}$) since we only want a Big-Oh bound.

The recursion tree has $\lg(n)$ height, each non-leaf node of the tree does $(\frac{n}{2^i})^2$ work, each leaf node does c_0 work, and each level has 2^i nodes.

So, the total work is $\sum_{i=0}^{\lfloor \lg(n) \rfloor - 1} 2^i \left(\frac{n}{2^i}\right)^2 + c_0 \cdot 2^{\lg n} = n^2 \sum_{i=0}^{\lfloor \lg(n) \rfloor - 1} \left(\frac{2^i}{4^i}\right) + c_0 n < n^2 \sum_{i=0}^{\infty} \left(\frac{1}{2^i}\right) + c_0 n = \frac{n^2}{1 - \frac{1}{2}} + c_0 n$.

This expression is upper-bounded by n^2 so $T \in \mathcal{O}(n^2)$.

3. Recurrences and Closed Forms

Consider the function g . Find a recurrence modeling the worst-case runtime of this function, and then find a closed form for the recurrence.

```
1 g(n) {
2   if (n <= 1) {
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  }
11  else {
12    for (int i = 0; i < n * n; i++) {
13      println("Yay!")
14    }
15    return 4 * g(n/3)
16  }
17 }
```

(a) Find a recurrence $T(n)$ modeling the worst-case time complexity of $g(n)$.

Solution:

$$T(n) = \begin{cases} c_0 & \text{if } n \leq 1 \\ 2T\left(\frac{n}{3}\right) + c_1n & \text{otherwise} \end{cases}$$

(b) Find a closed form for the above recurrence.

Solution:

The recursion tree has height $\log_3(n)$, each non-leaf level i has work $\frac{c_1n2^i}{3^i}$, and the leaf level has work $c_02^{\log_3(n)}$. Putting this together, we have:

$$\begin{aligned} \sum_{i=0}^{\log_3(n)-1} \left(\frac{c_1n2^i}{3^i}\right) + c_02^{\log_3(n)} &= c_1n \sum_{i=0}^{\log_3(n)-1} \left(\frac{2}{3}\right)^i + c_0n^{\log_3(2)} \\ &= c_1n \left(\frac{1 - \left(\frac{2}{3}\right)^{\log_3(n)}}{1 - \frac{2}{3}}\right) + c_0n^{\log_3(2)} && \text{By finite geometric series} \\ &= 3c_1n \left(1 - \left(\frac{2}{3}\right)^{\log_3(n)}\right) + c_0n^{\log_3(2)} \\ &= 3c_1n \left(1 - \frac{n^{\log_3(2)}}{n}\right) + c_0n^{\log_3(2)} \\ &= 3c_1n - 3c_1n^{\log_3(2)} + c_0n^{\log_3(2)} \end{aligned}$$

4. Runtime Complexity

Consider the function h :

```
1 h(n) {
2   if (n <= 1) {
3     return 1
4   } else {
5     return h(n/2) + n + 2*h(n/2)
6   }
7 }
```

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

Solution:

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

(b) Find a closed form to your answer for (a).

Solution:

The recursion tree has height $\lg(n)$, each non-leaf level i has work $c_1 2^i$, and the leaf level has work $c_0 2^{\lg(n)}$. Putting this together, we have:

$$\begin{aligned} \left(\sum_{i=0}^{\lg n - 1} c_1 2^i \right) + c_0 2^{\lg(n)} &= c_1 \left(\sum_{i=0}^{\lg n - 1} 2^i \right) + c_0 n = c_1 \frac{1 - 2^{\lg n - 1 + 1}}{1 - 2} + c_0 n \\ &= c_1 2^{\lg n} - c_1 + c_0 n \\ &= c_1(n - 1) + c_0 n \\ &= (c_0 + c_1)n - c_1 \end{aligned}$$