Recourrance and Big 'O' Notation, DSA pset2

Timothy Price

September 25, 2024

1 Pset2

1.1 Problem 1

```
for(int i=0; i < n - 5; ++i)
    for(int j=0; j < n/2; j++)
        Console.WriteLine($"i = {i} and j = {j});
```

The first for-loop runs $n-5$ times in total, so that averages to $O(n)$ time-complexity. The nested loop runs $n/2$ times everytime it is run so it also shares a time-complexity of $O(n)$. Because the loop is nested it runs $n/2$ times for every rotation the parent loop makes so we end up with a total time complexity of $O(n^2)$

1.2 Problem 2

```
for(int i=0; i < 2*n; ++i)
    for(int j=i * i; j > 0; j--)
        if( j > 2)
            Console.WriteLine($"{i} + {j} = {i+j}");
```

The parent for-loop runs a total of $2n$ times while the nested loop runs n^2 amount of times. The parent loop has a time complexity of $O(n)$ where the nested loop has a complexity of $O(n^2)$ which averages out to be $O(n^3)$

1.3 Problem 3

```
double x=1, y=n;
for(int i = 0; i <= 2*n; i++)
{
    x = 1;
    y = n;
    while(x<y)
    {
        x++;
        y--;
    }
}
```

The parent loop should run a total of $2n$ for a time-complexity of $O(n)$. The nested loop runs essentially $1/2n$ times and therefore has a time-complexity of $O(n)$, making the entire function's time complexity $O(n^2)$.

1.4 Problem 4

Here, we are going to expand the equation 3 times so that we understand what is happening when we nest operations inside of the function. We should start to notice a pattern that we can then generalize

$$\begin{aligned}
T(n) &= 2T(n-1) - 1 \\
T(0) &= 2 \\
T(n-1) &= 2((2T(n-1) - 1) - 1) - 1 \\
&= 2((2T(n-2) - 1) - 1 \\
&= (4T(n-2) - 2) - 1 \\
T(n-1) &= 4T(n-2) - 3
\end{aligned} \tag{1}$$

Once again, just substituting $T(n) = 2T(n-1) - 1$ for T(n-1) which just found equal to $4T(n-2) - 3$

$$\begin{aligned}
T(n-1) &= 4T(n-2) - 3 \\
T(n-2) &= 4T(2T(n-3) - 1) - 3 \\
&= 4T(2T(n-3) - 1) - 3 \\
&= 8T(n-3) - 4 - 3 \\
T(n-2) &= 8T(n-3) - 7
\end{aligned} \tag{2}$$

$$\begin{aligned}
T(n-2) &= 8T(n-3) - 7 \\
T(n-3) &= 8(2T(n-4) - 1) - 7 \\
&= 8(2T(n-4) - 1) - 7 \\
&= 16T(n-4) - 8 - 7 \\
T(n-3) &= 16T(n-4) - 15
\end{aligned} \tag{3}$$

So, from this I'm starting to notice that it looks like a sort of pattern is forming. Every nesting we do will result in something like: $2^k T(n-k) - (2^k - 1)$. Now that we have isolated the equation, we can actually substitute $T(0)$ to find our *base-case*. When $k = n$, we reach the *base-case*

$$\begin{aligned}
2^k T(n-k) - (2^k - 1) \\
2^n T(n-n) - (2^n - 1) \\
2^n T(0) - (2^n - 1) \\
2^n 2 - (2^n - 1) \\
2^{n+1} - 2^n - 1 \\
2^n(2-1) + 1 \\
2^n(1) + 1
\end{aligned} \tag{4}$$

After dropping the lower-order constants we are left with 2^n which translates to a time-complexity of $0(2^n)$.