Saleh Alkhalifa
9/27/2022

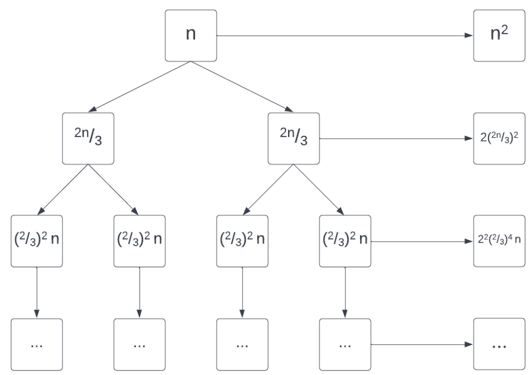
Homework#3

Problem 1 (5 points):

Solve the recurrence $T(n) = 2T\left(\frac{2}{3}\,n\right) + n^2$ first by using a recursion tree and then using the Master theorem. Show work.

Recursion Tree:

First, we calculate cost at each level:



$$\frac{2n}{3} + \frac{2n}{3} \rightarrow \left(\frac{2}{3}\right)^2 n^2 + \left(\frac{2}{3}\right)^2 n^2 \rightarrow 2\left(\frac{2}{3}n\right)^2$$

$$\left(\frac{2}{3}\right)^2 n^2 + \left(\frac{2}{3}\right)^2 n + \cdots \rightarrow 2^2\left(\frac{2}{3}\right)^4 n^2$$

Assuming k levels, at the kth level we see:

$$\left(\frac{2}{3}\right)^k n = 1, and\ so\ k = \ log_{2/3}n\ levels$$

Finally, identify asymptotic bound:

$$T(n) = \ n^2\left[\,2^0 * \left(\frac{2}{3}\right)^0 + 2^1 * \left(\frac{2}{3}\right)^1 + 2^2 * \left(\frac{2}{3}\right)^2 + \cdots to\ k\right]$$

$$T(n) = \ n^2\left[\left(\frac{8}{9}\right)^0 + \left(\frac{8}{9}\right)^1 + \left(\frac{8}{9}\right)^2 + \cdots to\ k\right]$$

$$T(n) = n^2\left[\frac{1 * \left(1 - \left(\frac{8}{9}\right)^{log_{\frac{3}{2}}n+1}\right)}{1 - \frac{8}{9}}\right] = n^2\left[\,1 - \frac{8}{9} * n^{log_{\frac{3}{2}}\left(\frac{8}{9}\right)}\,\right]$$

$$\therefore As\ n^{log_{\frac{3}{2}}\left(\frac{8}{9}\right)} \rightarrow 0, We\ see\ that\ T(n) = \theta(n^2)$$

Saleh Alkhalifa
9/27/2022

Homework#3

Master Theorem:

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

$$\ldots where\ f(n) is\ of\ the\ form\ n^c\ in\ which\ c \geq 0$$

$$T(n) \in \left\{\begin{matrix} if\ f(n) = O(n^{log_b a - \varepsilon}) & T(n) = O(n^{log_b a}) & Case\ 1 \\ if\ f(n) = \Omega(n^{log_b a + \varepsilon}) & T(n) = \theta(f(n)) & Case\ 2 \\ if\ f(n) = \theta(n^{log_b a} * (logn)^k) & T(n) = \theta(n^{log_b a} * (\log n)^{k+1}) & Case\ 3 \end{matrix}\right\}$$

$$First, we\ can\ rearrange:$$

$$T(n) = 2T\left(\frac{2}{3}\ n\right) + n^2$$

$$We\ see\ that\ f(n) = n^2, and\ n^{log_b a} = n^{log_{3/2} 2}$$

$$f(n) = \Omega(n^{log_b a + \varepsilon})$$

$$when\ f(n) \geq\ n^{log_{\frac{3}{2}} 2},$$

$$\therefore Therefore\ T(n) = \theta(n^2)$$

Saleh Alkhalifa
9/27/2022

Homework#3

Problem 2 (5 points):

Give asymptotic upper and lower bounds for the recurrence $T(n) = 3T\left(\frac{n}{2}\right) + \frac{n}{\log n}$. Justify your answer.

$$\textbf{\textit{Using the Master Theorem}}:$$

$$T(n) = aT\left(\frac{n}{b}\right) + f(n), where\ a \geq 1, b \geq 1, f\ is\ asymptotically\ positive$$

$$a = 3, b = 2, d = 1$$

$$\textbf{Case \#1}: \quad \boldsymbol{d > log_b a}$$

$$f(n) = \theta(n^d) \quad when\ d > log_b a$$

$$T(n)\ \epsilon\ \theta(f(n))$$

$$\textbf{Case \#2}: \quad \boldsymbol{d = log_b a}$$

$$f(n) = \theta(n^d \log n) \quad when\ d = log_b a$$

$$Therefore, T(n)\ \epsilon\ \theta(n^d \log n)$$

$$\textbf{Case \#3}: \quad \boldsymbol{d < log_b a}$$

$$f(n) = \theta(n^{log_b a}) \quad when\ d < log_b a$$

$$T(n)\ \epsilon\ \theta(n^{log_b a})$$

$$T(n) = 3T\left(\frac{n}{2}\right) + \frac{n}{logn}$$

$$When\ a = 3, b = 2, and\ f(n) = \frac{n}{\log n} < \text{n}$$

$$\therefore We\ can\ say\ that\ f(n) = \theta(n), and\ using\ case\ \#1, T(n)\ \epsilon\ \theta(n^{log_2 3})$$

Problem 3 (10 = 5 + 5 points):

Give asymptotic upper and lower bounds for each of the following recurrences. Justify your answer.
(a) $\boldsymbol{T(n) = \sqrt{n}T(\sqrt{n}) + n}$

$$T(n) = \sqrt{n} * T(\sqrt{n}) + n$$
$$T(n) = n^{\frac{1}{2}} * T\left(n^{\frac{1}{2}}\right) + n$$
$$T(n) = n^{\frac{1}{2}} * \left(n^{\frac{1}{4}} T\left(n^{\frac{1}{4}} + n^{\frac{1}{2}}\right)\right) + n$$
$$T(n) = n^{\frac{3}{4}} * T\left(n^{\frac{1}{2}}\right) + 2n$$
$$T(n) = n^{\frac{7}{8}} * T\left(n^{\frac{1}{8}}\right) + 3n$$
$$T(n) = n^{1-\frac{1}{2}^{k}} * T\left(n^{\frac{1}{2}^{k}}\right) + kn$$
$$\textit{In the event that } n^{\frac{1}{2}^{k}} \textit{ is less than } 2, \textit{then } k > \log\log n$$
$$\therefore \mathrm{T}here fore, T(n) = \theta(n \log\log n)$$

$$\textit{Alternatively we can also say that}$$
$$S(n) = \frac{T(n)}{n}$$
$$\textit{In this case, the recurrence now becomes:}$$
$$S(n) = S(\sqrt{n}) + n$$
$$\textit{Via recursion tree, we can see that } S(n) = \theta(n \log\log n)$$
$$\therefore \mathrm{T}here fore, T(n) = \theta(n \log\log n)$$

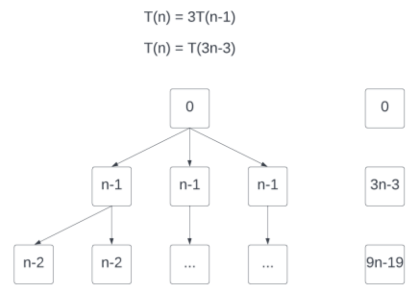
(b) $\boldsymbol{T(n) = 3T(n-1)}$

$$\textit{Via Recursion Tree, we can say that:}$$

$$T(n) = 3T(n-1)$$

$$T(n) = 3T(n-1) + 0$$

$$T(n) = T(3n-3) + 0$$
$$= 3^0 n + 3^1 n + 3^2 n + 3^3 n + \cdots$$
$$\therefore \text{Therefore}, T(n) = \theta(3^n)$$



$$\textit{In addition:}$$

$$T(n) = 3(3T(n-2)$$

$$T(n) = 3^2 T(n-2)$$

$$T(n) = 3^2\big(3T(n-3)\big) = 3^3\big(T(n-3)\big)$$

$$\textit{Given a value } k\textit{, we can say that}$$

$$T(n) = 3^k\big(T(n-k)\big)$$

$$T(n) = 3^n\big(T(0)\big)$$

$$\therefore \textit{Therefore}, T(n) = \theta(3^n)$$

Problem 4: Programming Assignment (10 points):

- **Maximum Product Subarray**
- **Given an integer array `nums`, find a contiguous non-empty subarray within the array that has the largest product, and return *the product*.**
- **The test cases are generated so that the answer will fit in a 32-bit integer.**
- **A subarray is a contiguous subsequence of the array.**

Please find a zip file attached with the code provided in the form of a Jupyter notebook. Using **Python3**, you can install Jupyter Notebook via 'pip install jupyter notebook'. Once installed, you can run the commands you will see below.

The following figure shows the function **maximumSubArrayProductFinder** which takes one argument, an array **nums**. The function will check for an empty or too large of an array, before going through the given array to find the largest product of a sub array.

```
[47]: def maximumSubArrayProductFinder(nums):
    """
    Returns the largest product of a subarray found in
    the input array nums

    Input (array) : An array such as [1, 2, 3, -5]

    Returns (int) :  An integer representing the largest product of a subarray
    """

    # Check for empty array
    if len(nums) < 1:
        return "Empty Array"
    elif len(nums) > 20000:
        return "Too large"

    # Reverse order of array
    nums_reversed = nums[::-1]

    print("Original: ", nums)
    print("Reversed: ", nums_reversed)

    # Iterate over the array nums
    for num in range(1, len(nums)):

        # Alter nums[num] with the product
        nums[num] *= nums[num - 1] or 1

        # Alter nums_reversed[num] with the product
        nums_reversed[num] *= nums_reversed[num - 1] or 1

    # Complete array:
    full_array = nums + nums_reversed

    # Find maximum:
    maximum = max(full_array)

    # Return the maximum of the array
    return maximum
```

Homework#3

The following figure show 11 test cases developed to demonstrate the both the examples included in the assignments specification, as well as a few others. Each test case has an **input_array_n**, the **function** being executed with that array, as well as the **output**. These test cases show the utility of the application to handle the ability of finding sub array products, as well as ensuring that edge cases are handled properly.

```
input_array_1 = [2,3,-2,4]
maximumSubArrayProductFinder(input_array_1)
```

```
Original:  [2, 3, -2, 4]
Reversed:  [4, -2, 3, 2]
6
```

```
input_array_2 = [-2, 0, -1]
maximumSubArrayProductFinder(input_array_2)
```

```
Original:  [-2, 0, -1]
Reversed:  [-1, 0, -2]
0
```

```
input_array_3 = [-10, 5, -5, 0, 10]
maximumSubArrayProductFinder(input_array_3)
```

```
Original:  [-10, 5, -5, 0, 10]
Reversed:  [10, 0, -5, 5, -10]
250
```

```
input_array_4 = [-1, 2, -3, 4, 5]
maximumSubArrayProductFinder(input_array_4)
```

```
Original:  [-1, 2, -3, 4, 5]
Reversed:  [5, 4, -3, 2, -1]
120
```

```
input_array_5 = [-10, 5, -5, 0, 10, 2, 5, 7, 10, 8, 5, 1, -10]
maximumSubArrayProductFinder(input_array_5)
```

```
Original:  [-10, 5, -5, 0, 10, 2, 5, 7, 10, 8, 5, 1, -10]
Reversed:  [-10, 1, 5, 8, 10, 7, 5, 2, 10, 0, -5, 5, -10]
280000
```

```
input_array_6 = [-2, 0, 0, 0, 0, -55]
maximumSubArrayProductFinder(input_array_6)
```

```
Original:  [-2, 0, 0, 0, 0, -55]
Reversed:  [-55, 0, 0, 0, 0, -2]
0
```

```
input_array_7 = [-2, -55, 0]
maximumSubArrayProductFinder(input_array_7)
```

```
Original:  [-2, -55, 0]
Reversed:  [0, -55, -2]
110
```

```
input_array_8 = [0, -2, -1]
maximumSubArrayProductFinder(input_array_8)
```

```
Original:  [0, -2, -1]
Reversed:  [-1, -2, 0]
2
```

```
input_array_9 = [0]
maximumSubArrayProductFinder(input_array_9)
```

```
Original:  [0]
Reversed:  [0]
0
```

```
input_array_10 = []
maximumSubArrayProductFinder(input_array_10)
```

```
'Empty Array'
```

```
input_array_11 = [5]*20001
maximumSubArrayProductFinder(input_array_11)
```

```
'Too large'
```