Student: Dakin Werneburg
Section 6381

Instructor: Susan Furtney
Date: January 25, 2019

CMSC 451 Homework 2

1. Given the following two functions:

- ***f*(*n*) = 3*n*^{2} + 5**
- ***g*(*n*) = 53*n* + 9**

Use L'Hopital's rule and limits to prove or disprove each of the following:

- ***f*** $\in$ **Ω(*g*)**
- ***g*** $\in$ **Θ(*f*)**

Solution:

Big O Definition: $f(n) = O(g(n))$ $if\ there\ exist\ some\ constant\ C > 0\ and\ n_0 \geq 1, such\ that\ f(n) \leq Cg(n) for\ every\ n \geq\ n_0$

Big Ω Definition: $f(n) = \Omega(g(n))$ $if\ there\ exist\ some\ constant\ C > 0\ and\ n_0 \geq 1, such\ that\ f(n) \geq Cg(n) for\ every\ n \geq\ n_0$

Big Θ Definition: $f(n) = \Theta(g(n))\ \ if\ f(n) = O(g(n)) and\ f(n) = \Omega(g(n))$

$$\lim_{n\to\infty}\frac{f(n)}{g(n)} \to L'Hopital\ \lim_{n\to\infty}\frac{f'(n)}{g'(n)} \quad \Longrightarrow \quad \lim_{n\to\infty}\frac{3n^2+5}{53n+9} \quad = \quad \lim_{n\to\infty}\frac{6n}{53} \quad = \quad \infty$$

$$\lim_{n\to\infty}\frac{g(n)}{f(n)} \to L'Hopital\ \lim_{n\to\infty}\frac{g'(n)}{f'(n)} \quad \Longrightarrow \quad \lim_{n\to\infty}\frac{53n+9}{3n^2+5} \quad = \quad \lim_{n\to\infty}\frac{53}{6n} \quad = \quad 0$$

$$Therefore: \quad 0 \geq g(n) \geq f(n) \geq \infty$$

$f \in\ \Omega(g)$
$g \in\ \Theta(f)$

both are true statements

Student: Dakin Werneburg | Instructor: Susan Furtney
Section 6381 | Date: January 25, 2019

2. Rank the following functions from lowest asymptotic order to highest. List any two or more that are of the same order on the same line.

- **2^n**
- **n^3+5n**
- **$\log_2 n$**
- **n^3+2n^2+1**
- **3^n**
- **$\log_3 n$**
- **$n^2+5n+10$**
- **$n\log_2 n$**
- **$10n+7$**
- **$\sqrt{n}$**

Solution:

- $log_2 n,\ \ log_3 n$
- $10n + 7$
- $nlog_2 n$
- $\sqrt{n}$
- $n^2 + 5n + 10$
- $n^3 + 5n,\ n^3 + 2n^2 + 1$
- $2^n,\ 3^n$

Student: Dakin Werneburg
Section 6381

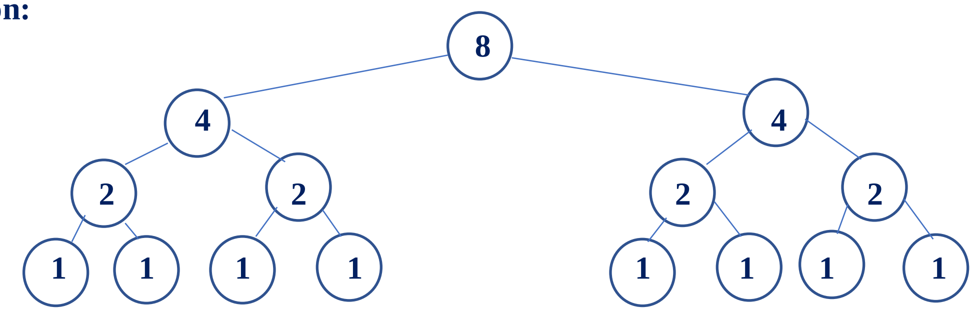
Instructor: Susan Furtney
Date: January 25, 2019

3. Draw the recursion tree when *n* = 8, where *n* represents the length of the array, for the following recursive method:

```
int sum(int[] array, int first, int last)
{
  if (first == last)
    return array[first];
  int mid = (first + last) / 2;
  return sum(array, first, mid) + sum(array, mid + 1, last);
}
```

- **Determine a formula that counts the numbers of nodes in the recursion tree.**
- **What is the Big-Θ for execution time?**
- **Determine a formula that expresses the height of the tree.**
- **What is the Big-Θ for memory?**
- **Write an iterative solution for this same problem and compare its efficiency with this recursive solution.**

Solution:



- The formula for the number of nodes in the tree is: $t(n) = 2n - 1$
- Big Θ execution time = $\Theta(n)$
- Height of Tree = $log_2 n$
- Big Θ for memory = $\Theta(log_2 n)$
- The iterative version does have the same time complexity asymptotically with an order of O(n) but requires more memory than the recursive which needs O(log n) vs O(n).

```
int sum (int [] array ){
   int sum = 0;
   for (int i=0; i< array.lenght; i++{
      sum += array[i];
    }
   return sum;
}
```

Student: Dakin Werneburg
Section 6381

Instructor: Susan Furtney
Date: January 25, 2019

4. Using the recursive method in problem 3 and assuming *n* is the length of the array.

- **Modify the recursion tree from the previous problem to show the amount of work on each activation and the row sums.**
- **Determine the initial conditions and recurrence equation.**
- **Determine the critical exponent.**
- **Apply the Little Master Theorem to solve that equation.**
- **Explain whether this algorithm optimal.**

There is constant amount of non-recursive work each call, therefore

Level		work at each level
0	$T\left(\frac{n}{2^0}\right)$	$C*2^0$
1	$T(\frac{n}{2^1})$ $T(\frac{n}{2^1})$	$C*2^1$
2	$T(\frac{n}{2^2})$ $T(\frac{n}{2^2})$ $T(\frac{n}{2^2})$ $T(\frac{n}{2^2})$	$C*2^2$
3	$T(\frac{n}{2^3})$ $T(\frac{n}{2^3})$ $T(\frac{n}{2^3})$ $T(\frac{n}{2^3})$ $T(\frac{n}{2^3})$ $T(\frac{n}{2^3})$ $T(\frac{n}{2^3})$ $T(\frac{n}{2^3})$	$C*2^3$
i	$\frac{n}{2^i}\ldots$	$C*2^i$

Total. $C\left(\frac{1-2^{log_2 n+1}}{1-2}\right) = C\left(\frac{1-2n}{-1}\right) = C(2n-1)$

Initial Condition: T(n) = 1, for n<=1

Recurrence Equation: T(n) = 2T(n/2) + O(1). for n > 1

Critical Exponent: E = $\log_b a$ = $\log_2 2$ = 1

Master Theorem: If $T(n) = aT\left(\frac{n}{b}\right) + O(n^d)$ for constants a>0, b>1, d>= 0, then:.

$$T(n) = \begin{cases} O(n^d), & if\ d > log_b a \\ O(n^d \log n, & if\ d = log_b a \\ O(n^{log_b a}), & if\ d < log_b a \end{cases}$$

$O(1) = O(n^0)$ $a = 2,\ \ b = 2\ \ \ d = 0$ $0 < log_2 2$

Therefore: $O(n^{log_b a}) = O(n^{log_2 2}) = O(n)$

Optimal? This algorithm is optimal when it comes to space complexity as an order of $log_2 n$ but has the same time complexity with an order of $O(n)$ in both the iterative and recursive functions. Additionally, there could be no optimal way to improve the time complexity of O(n) because we would still need to access every element of the array.