**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*  ∈ Ω(*g*)

**true if: *lim n→∞ f/g ≠ 0***

**lim n→∞ f(n)/g(n) = lim n→∞ (3n2 + 5) / (53n + 9)**

**take the derivative (applying L’Hopital)**

**lim n→∞ 6n / 53 = ∞**

**Therefore, *f*  ∈ Ω(*g*) is TRUE**

* *g* ∈ Θ(*f*)

**true if: *lim n→∞ g/f ≠ ∞, 0***

**lim n→∞ g(n)/f(n) = lim n→∞ (53n + 9) / (3n2 + 5)**

**take the derivative (applying L’Hopital)**

**lim n→∞ 53/6n = 0**

**Therefore, g ∉ Θ(*f*) and *g* ∈ Θ(*f*)** **is FALSE**

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.

* **log3 𝑛, log2𝑛 (base irrelevant)**
* **√𝑛**
* **10𝑛+ 7**
* **𝑛 log2 𝑛**
* **𝑛2+5𝑛+10**
* **𝑛3+2𝑛2+1, 𝑛3+5𝑛 (as n grows, only n3 remains relevant)**
* **2𝑛**
* **3n**

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) {

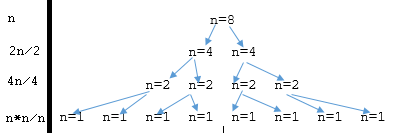
if (first == last)

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.

**2n – 1, where n >= 1**

**This works assuming we are using integer division, i.e.: 7/2 = 3 (not 4 or 3.5)**

**Otherwise, we need to use logarithms for the equation: 2log2n+1 – 1**

**Test with n=8: 2log2n+1– 1 = 2log28 + 1– 1 = 23+1 – 1 = 24 – 1 = 16 -1 = 15**

**or… 2(8) – 1 = 16 -1 = 15**

* What is the Big-Θ for execution time?

**T(n) = 2\*T(n/2) + 1 … with master thm, a=2, b=2, f(n) = 1 … = Θ(nlog22) = Θ(n)**

* Determine a formula that expresses the height of the tree.

**h = log2n + 1**

* What is the Big-Θ for memory?

**This is a function of the height of the recursion tree,** **h = log2n + 1**

**h(n) = log2n + 1, therefore, h ∈ Θ(log n)**

* Write an iterative solution for this same problem and compare its efficiency with this recursive solution.

int sum(int[] array, int first, int last) {

int result = 0;

for (int i = first; i <= last; i++) {

result += array[i];

}

return result;

}

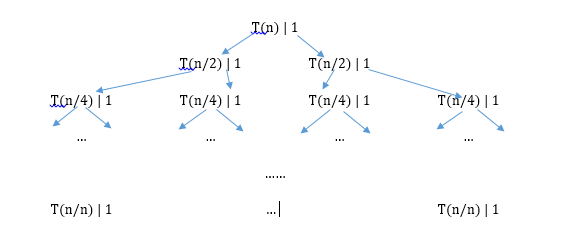
**The for loop is what will be affected by n or input size. The for loop will run n times and so the time complexity of the iterative method is also Θ(n).**

**The iterative will use less space than the recursive algorithm as it is Θ(1) since *result* is the only memory space used at a high level (not accounting for how machines do arithmetic with registers – not that is should matter as the memory space used will not grow as n grows.)**

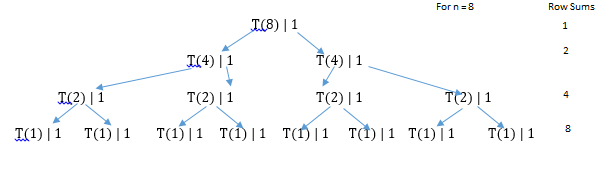
4. Using the recursive method in problem 3 and assuming *n* is the length of the array.

* Modify the recursion tree from the previous problem to show the amount of work on each activation and the row sums.

**Recursion tree if n is unknown:**



**If n = 8 as in previous problem:**



* Determine the initial conditions and recurrence equation.

Initial condition: **T(1) = 1**

Recurrence equation: **T(n) = 2T(n/2) + 1**

* Determine the critical exponent.

**Since row of sums is increasing, T ∈ Θ(nE), E is critical exponent**

**E = lg(b)/lg(c) for T(n) = bT(n/c) + f(n)**

**Plugging the numbers in from our recurrence equation: T(n) = 2T(n/2) + 1**

**So, b=2 and c=2**

**E = lg(2)/lg(2) = 1**

* Apply the Little Master Theorem to solve that equation.

**T(n) ∈ Θ(nE), Since E = 1: T(n) ∈ Θ(n)**

* Explain whether this algorithm optimal.

**This algorithm is optimal at Θ(n) since every input element needs to be inspected in order to perform the operations to set the variable, middle. There is no way to operate on each element with time better than Θ(n).**