Homework 2 sample solution

Due 09/17/2024

September 17, 2024

Analyze the worst-case time complexity of the algorithms below. You may express their complexity using Big-Oh or Big-Theta. If you use Big-Oh, you should give a *tight* upper bound on the complexity (the smallest possible valid bound). For example, if an algorithm has a complexity of $\Theta(n^3)$, then $O(n^3)$ is a tight upper bound, but $O(n^4)$ is not.

Show all work.

1.

```
   Input: n: positive integer
   Input: k: positive integer
1  Algorithm: LoopMystery1
2  ret = 0
3  for i = 1 to n do
4  |   for j = i to n step k do
5  |   |   for ℓ = 0 to k do
6  |   |   |   ret = ret + i(j + ℓ)
7  |   |   end
8  |   end
9  end
10 return mystery
```

Lines 2, 6, and 10 are $\Theta(1)$. The inner loop (line 5) iterates k times, and each iteration takes $\Theta(1)$ time, for a total of $\Theta(k)$ time. The middle loop (line 4) iterates $\frac{n-i}{k}$ times. Each iteration takes $\Theta(k)$ time (which is not changing), so the total time for the middle loop is $\frac{n-i}{k}\Theta(k) = \Theta(n-i)$. The outer loop (line 3) iterates n times at $\Theta(n-i)$ per iteration, for a total of $\sum_{i=1}^{n} n-i = (n-1)+(n-2)+\ldots+1 = 1+2+\ldots+(n-1) = \sum_{i=1}^{n-1} i = \Theta(n^2)$. Since lines 2 and 10 take $\Theta(1)$, LoopMystery takes $\Theta(n^2)$ time total.

2.

```
   Input: n: positive integer
1  Algorithm: LoopMystery2
2  ret = 0
3  i = 1
4  max = n · n · n
5  while i ≤ max do
6  |  ret = ret + i
7  |  i = 2i
8  end
9  return ret
```

Lines 2, 3, 4, 6, 7, and 9 all take $\Theta(1)$ time. The loop in line 5 is controlled by the variable i, which starts at 1 and doubles each iteration until it is greater than n^3. This will take $lg(n^3) = 3\lg n = \Theta(\lg n)$ iterations. Each iteration of the loop takes $\Theta(1)$ time, so the total time for the loop (and therefore for LoopMystery2) is $\Theta(\lg n)$.

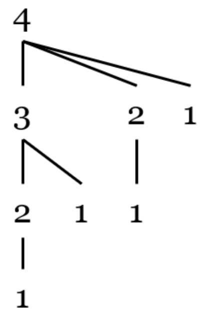
3. Answer the following questions about the worst-case complexity of the recursive algorithm below.

```
    Input: n: positive integer
1   Algorithm: RecursionMystery
2   if n = 1 then
3   |  return 1
4   else
5   |  ret = 0
6   |  for i = 1 to n − 1 do
7   |  |  ret = ret + RecursionMystery(i)
8   |  end
9   |  return ret
10  end
```

(a) What is the *nonrecursive* complexity of a single call to RecursionMystery with an input of size k?

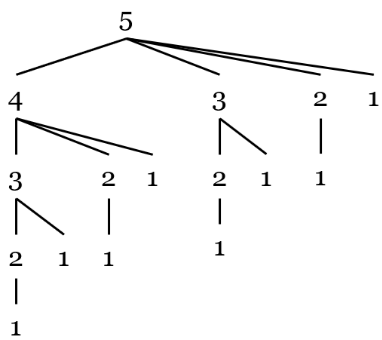
$\Theta(n)$: lines 2–5, 7, and 9 are all $\Theta(1)$. The loop iterates $\Theta(n)$ times at $\Theta(1)$ per iteration (ignoring the recursive calls), for a total of $\Theta(n)$ time, which dominates the lines outside of the loop.

(b) Sketch a recursion tree for $n = 4$. How many recursive calls are there of size 1, 2, 3, and 4, respectively?



Size	Count
4	1
3	1
2	2
1	4

(c) Sketch a recursion tree for $n = 5$. How many recursive calls are there of size 1, 2, 3, 4, and 5, respectively?



Size	Count
5	1
4	1
3	2
2	4
1	8

(d) Write a summation that approximates the time complexity for RecursionMystery(n). You do not need to simplify this summation.

The complexity of RecursionMystery is given by the summation:

$$\Theta\left(n + \sum_{i=1}^{n-1} i2^{n-1-i}\right)$$

(It is okay if the summation is slightly off, like changing the bound to be 1 to n or using 2^{n-i} instead of 2^{n-1-i}.)

If you examine the recursion tree, there is one recursive call of size n (taking $\Theta(n)$ time), one call of size $n-1$ (taking $\Theta(n-1)$ time), and there are twice as many of every subsequent call: two of size $n-2$, four of size $n-3$, eight of size $n-4$, and so forth.

It is possible to show that the overall complexity adds up to $\Theta(2^n)$; however, this was not asked.