Student: John Rizzo

Course: CS590-A Algorithms

Instructor: Dr. William Hendrix

Due Date: February 17, 2025

Description: Homework 2 Algorithms

Problem 1

Analyze the *worst-case* time complexity of the LoopMystery algorithm below. Please show all work. The $\lfloor\rfloor$ symbols represent the *floor* ("round down") function. You may assume that this function takes $\Theta(1)$ time for any input. You may also assume it takes a constant amount of time to determine whether an integer is odd.

Note that figuring out what problem this algorithm solves is *irrelevant* to analyzing its complexity.

```
Input   : n: nonnegative integer
1 Algorithm: LoopMystery
2 sum = 0
3 t = 1
4 d = 1
5 k = n
6 while k > 1 do
7     for i = 1 to k do
8         t = t + d
9         sum = sum + t
10    end
11    if k is odd then
12        d = -d
13    end
14    k = ⌊k/2⌋
15 end
16 return sum
```

1. lines 1-5 are simple constant time assignment operations
2. lines 7-10 is the first inner loop with lines 8, 9 being constant time and since $k = \lfloor k/2 \rfloor$ using the ... approach if you choose you can see that
   $(n-1) + \frac{(n-1)}{2} + \frac{(n-1)}{4}$.
   $(n-1)(\frac{1}{2} + \frac{1}{4})$

For large numbers of n it will dominate the fraction. Based on this sequence you can see that this is a sum of iterations that matches $T(n) = \sum_{i=1}^{n} \frac{1}{2^i} = \Theta(1)$

3. lines 11-13 is a simple conditional which should be constant time.

4. lines 6-16 is the largest outter loop will be dominated by the first inner loop

Answer $T(n) = \sum_{i=1}^{n} \frac{1}{2^i} = \Theta(1)$

Problem 2

Find a recurrence $T(n)$ that describes the runtime of the RecursionMystery algorithm below:

```
Input   : data: array of integers
Input   : n: size of data
01 Algorithm: RecursionMystery
02 if n > 1 then
03     min = max = 1
04     for i = 2 to n do
05         if data[i] < data[min] then
06             min = i
07         end
08         if data[i] > data[max] then
09             max = i
10         end
11     end
12     Swap data[1] and data[min]
13     if max > 1 then
14         Swap data[n] and data[max]
15     else
16         Swap data[min] and data[max]
17     end
18     if n > 2 then
19         Call RecursionMystery on data[2..n − 1]
20     end
21 end
22 return data
```

1. lines 1-3,12-17,21-22 are constant time

2. lines 4-11 is a non-recursive loop which will occur a max of $n-1$ times

3. lines 19 has the only recursive call which will occur a maximum of n-2 times. For example if $n = 6$ then
2..6-1, 2..6-2, 2..6-3 and so on. This matches $\sum_{i=l}^{n} i = \Theta(x^2)$

Answer $T(n) = \Theta(x^2)$

Problem 3

Sketch the recurrence tree that corresponds to the recurrence $T(n) = 4T(\frac{n}{2}) + \Theta(1)$

$T(\frac{n}{2})$ $\Theta(1)$

$\frac{n}{4}$ $\frac{n}{4}$ $2\Theta(1)$

$\frac{n}{8}$ $\frac{n}{8}$ $\frac{n}{8}$ $\frac{n}{8}$ $4\Theta(1)$

and so on...

Height: $2^n\Theta(1)$

Total Complexity: $\sum_{i=1}^{n}\Theta(1)$

At each level it most closely match a multiple of $\Theta(1)$

Problem 4

Find a recurrence that describes the worst-case complexity of the Third-Sort algorithm below. Show all work. You may assume that the floor function ($\lfloor\ \rfloor$) takes constant time.

Input : data: array of integers
Input : n: the length of data
Output a permutation of data such that $data[1] \leq data[2] \leq ... \leq data[n]$
c **Algorithm**: ThirdSort
c **if** $n = 1$ **then**
c   **return** *data*
c **else if** $n = 2$ **then**
c   **if** $data[1] > data[2]$ **then**
c   Swap $data[1]$ and $data[2]$
c   **end**
c   **return** data
c **else**
c $third = \lfloor\frac{n}{3}\rfloor$
T$(\frac{2n}{3})$ Call ThirdSort on $data[1..n - third]$
T$(\frac{2n}{3})$ Call ThirdSort on $data[third + 1..n]$
T$(\frac{2n}{3})$ Call ThirdSort on $data[1..n - third]$
c   **return** *data*
c **end**

$T(n) = 3T(\frac{2n}{3})) + f(n)$ which in this case f(n) occurs n times which is $O(1)$

Answer $T(n) = 3T(\frac{2n}{3}) + O(1)$

Problem 5

Use the Master Theorem to find the worst-case complexity of ThirdSort. You may assume that $f(n)$ is regular if relevant. Recall that $log_a(b) = \frac{ln(b)}{ln(a)}$ (you may need a calculator for this one). Be sure to include the value of c and the case of the Master Theorem in your answer.

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

$\frac{n}{b}$ is the size of recursive calls
a is the number of recursive calls, possibly equal to b
$f(n)$ is the time for other code
From Problem 4, $T(n) = 3\Theta(\frac{2n}{3})$, which means $a = 3$ and $b = \frac{3}{\frac{1}{2}} = 3 \times 0.5 = 1.5$
$c = log_b(a) = \frac{ln(b)}{ln(a)} = log_{1.5}(3) = \frac{ln(1.5)}{ln(3)} = 0.37$

Answer

Since $f(n) = O(1)$ and $n^c = n^{0.37}$, $f(n) < n^c$ for large n, therefore $\Theta(n^c)$