CSCE 222 (Carlisle), Discrete Structures for Computing
Spring 2022
Homework 6

Type your name below the pledge to sign
On my honor, as an Aggie, I have neither given nor received unauthorized aid on this academic work.
HUY QUANG LAI

Instructions:

- The exercises are from the textbook. You are encouraged to work extra problems to aid in your learning; remember, the solutions to the odd-numbered problems are in the back of the book.
- Grading will be based on correctness, clarity, and whether your solution is of the appropriate length.
- Always justify your answers.
- Don't forget to acknowledge all sources of assistance in the section below, and write up your solutions on your own.
- *Turn in .pdf file to Gradescope by the start of class on Monday, February 28, 2022.* It is simpler to put each problem on its own page using the LaTeX clearpage command.

Help Received:

- Rosen, Kenneth H. *Discrete Mathematics and Its Applications*. McGraw-Hill, 2019.

Exercises for Section 3.2:

6: (2 points)
Show that $\dfrac{x^3+2x}{2x+1}$ is $O(x^2)$
To satisfy the definition of $O(x^2)$, we need to find an appropriate choice of c and k.
Let's consider $x \geq 1000$ and reason by inequalities:

$$\frac{x^3+2x}{2x+1} < \frac{x^3+2x}{2x} < \frac{1}{2}x^2 + x^2 = \frac{3}{2}x^2$$

Since $\dfrac{x^3+2x}{2x+1} < \dfrac{3}{2}x^2$ when $x \geq 1000$, we can choose $k = 1000$ and $c = \frac{3}{2}$

22: (2 points)
Arrange the functions $(1.5)^n, n^{100}, \log^3 n, \sqrt{n}\log n, 10^n, (n!)^2,$ and $n^{99}+n^{98}$ in a list so that each function is big-O of the next function.
$\log^3 n, \sqrt{n}\log n, n^{99}+n^{98}, n^{100}, (1.5)^n, 10^n, (n!)^2$

26(a-c): (2 points)
Give a big-O estimate for each of these functions. For the function g in your estimate $f(x)$ is $O(g(x))$, use a simple function g of smallest order.

1. $(n^3 + n^2\log n)(\log n + 1) + (17\log n + 19)(n^3 + 2)$
   $O(n^3 \log n)$
2. $(2^n + n^2)(n^3 + 3^n)$
   $O(6^n)$
3. $(n^n + n2^n + 5^n)(n! + 5^n)$
   $O(n^n \cdot n!)$

74: (2 points)
Determine whether $\log n!$ is $\Theta(n \log n)$. Justify your answer.
$\log n! = \log 1 + \log 2 + \log 3 + \cdots + \log n$

Upper Bound:
$\log 1 + \log 2 + \log 3 + \cdots + \log n \leq \log n + \log n + \log n + \cdots \log n$
$= n \cdot \log n$

Lower Bound:
$\log 1 + \cdots + \log \dfrac{n}{2} + \cdots + \log n \geq \log \dfrac{n}{2} + \log\left(\dfrac{n}{2}+1\right) + \cdots + \log(n-1) + \log n$
$= \dfrac{n}{2} \cdot \log\left(\dfrac{n}{2}\right)$

Exercises for Section 3.3:

2: (1 point).
Give a big-O estimate for the number additions used in this segment of an algorithm.

```
t:=0
for i:=1 to n
    for j:=1 to n
        t:=t+i+j
```

$O(n^2)$

4: (1 point).
Give a big-O estimate for the number of operations, where an operation is an addition or a multiplication, used in this segment of an algorithm (ignoring comparisons used to test the conditions in the while loop).

```
i := 1
t := 0
while i <= n
    t := t + 1
    i := 2i
```

Addition: $O(\log n)$
Multiplication: $O(\log n)$

8: (2 points).
Given a real number x and a positive integer k, determine the number of multiplications used to find x^{2^k} starting with x and successively squaring (to find x^2, x^4, and so on). Is this a more efficient way to find x^{2^k} than by multiplying x by itself the appropriate number of times?
Successively squaring will double the exponent of x^2 k number of times. Because of this, successively squaring would be $O(k)$.
This is more efficient than multiplying x 2^k number of times as the number of multiplication which is $O(2^k)$

12b: (2 points).

Consider the following algorithm, which takes as input a sequence of n integers $a_1, a_2, \cdots, a_n$ and produces as output a matrix $M = \{m_{ij}\}$ where m_{ij} is the minimum term in the sequence of integers $a_i, a_{i+1}, \cdots, a_j$ for $j \geq i$ and $m_{ij} = 0$ otherwise.

initialize M so that $m_{ij} = a_i$ if $j \geq i$ and $m_{ij} = 0$ otherwise
for i := 1 to n.
    for j := i + 1 to n.
        for k := i + 1 to j
            m_{ij} := min(m_{ij}, a_k)
return $M = m_{ij}$ $\{m_{ij}$ is the minimum term of $a_i, a_{i+1}, \cdots, a_j\}$

Show that this algorithm uses $\Omega(n^3)$ comparisons to compute the matrix M. Using this fact and part (a), conclude that the algorithms uses $\Theta(n^3)$ comparisons. [Hint: Only consider the cases where $i \leq \frac{n}{4}$ and $j \geq \frac{3n}{4}$ in the two outer loops in the algorithm.]

The outermost loop (i) will run more than $\frac{n}{4}$ times. In other words, $i \geq \frac{n}{4}$ Because of this, the second nested loop (j) will run at least $1 - i$ times. In other words, $j \geq \frac{3n}{4}$. A similar argument can be applied to the innermost loop k to get $k \geq \frac{3n}{4}$. Because of this, the algorithm will loop more than $i \times j \times k$ times.

$f(x) \geq \dfrac{n}{4} \cdot \dfrac{3n}{4} \cdot \dfrac{3n}{4}$

$f(x) \geq \dfrac{9}{64} n^3$

Because of this, the algorithm is $\Omega(n^3)$.

Since the algorithm is both $O(n^3) \wedge \Omega(n^3)$, the algorithm must also be $\Theta(n^3)$

14a: (1 points)

There is a more efficient algorithm (in terms of the number of multiplications and additions used) for evaluating polynomials than the conventional algorithm described in the previous exercise. It is called **Horner's method**. This pseudocode shows how to use this method to find the value of $a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$ at $x = c$.

procedure Horner($c, a_0, a_1, a_2, \cdots, a_n$: real numbers)
$y := a_n$
for $i := 1$ to n
    $y := y \cdot c + a_{n-i}$
return $y\{y = a_n c^n + a_{n-1} c^{n-1} + \cdots + a_1 c + a_0\}$

Evaluate $3x^2 + x + 1$ at $x = 2$ by working through each step of the algorithm showing the values assigned at each assignment step.

$y := 2$ Initial Condition

$y := 2 \cdot 2 + 1 = 7$ $i = 1$

$y := 7 \cdot 2 + 1 = 15$ $i = 2$ return 15

$3x^2 + x + 1$ evaluated at $x = 2$ is 15.

14b: (1 points)
Exactly how many multiplications and additions are used by this algorithm to evaluate a polynomial of degree n at $x = c$? (Do not count additions used to increment the loop variable.)
n multiplications and n additions.
One each in step which we do n times in the for-loop.

20(b,c,e,g): (2 points)

What is the effect in the time required to solve a problem when you double the size of the input from n to $2n$, assuming that the number of milliseconds the algorithm uses to solve the problem with input size n is each of these functions? [Express your answer in the simplest form possible, either as a ratio or a difference. Your answer may be a function of n or a constant.]

$\log n$

$$\begin{aligned} &\log 2n - \log n \\ =&\log \frac{2n}{n} \\ =&\log 2 \end{aligned}$$

n^2

$$\frac{(2n)^2}{n^2}$$
$=4$

$100n$

$$\frac{100(2n)}{100n}$$
$=2$

2^n

$$\frac{2^{2n}}{2^n}$$
$=2^n$

42: (2 points)

Find the complexity of the greedy algorithm for scheduling the most talks by adding at each step the talk with the earliest end time compatible with those already scheduled (Algorithm 7 in Section 3.1). Assume that the talks are not already sorted by earliest end time and assume that the worst-case time complexity of sorting is $O(n \log n)$.

The first step of the greedy algorithm would be to sort the lest by end-time. This would take, at most, $O(n \log n)$.
Then the greedy algorithm would traverse the list backwards to greedily schedule meetings by end-time. This process would take, at most, $O(n)$.
Therefore, the algorithm as a whole would take, at most, $O(n \log n + n)$ which can be simplified to $O(n \log n)$.