CS220 Discrete Math - Homework #4

Brendan Nguyen - brendan.nguyen001@umb.edu

February 24, 2022

Question 1

The definition of big-*Onotation* tells you that have you to find the witnesses C and k such that $f(x) \leq C(g(x))$ when $f(x) = O(g(x))$.

First, I tested $c = 2$ and $k = 2$. Testing the definition for $n = 3$, I got $f(2.5) \leq c(g(2.5)) \rightarrow 25 \leq 54$. Although this pair seemed to work, when testing values approaching $n = 2$, I found that $n = 2.1$ fails the inequality ($f(2.1) = 21.287$ and $2(g(2.1)) = 20.090$).

Afterwards, I tested $c = 2$ and $k = 3$. Plugging in $n = 4$, I got $f(4) \leq c(g(4)) \rightarrow 33 \leq 162$. Testing values approaching $n = 3$ shows us that the **witnesses** $c = 2$ and $k = 3$ are valid as no decimals fail the inequality.

Question 2

According to the definition of big-O notation, we can say that:

$$1^k + 2^k + \cdots + n^k < n^k + n^k + \cdots + n^k = n \times n^k = n^{k+1}$$

Since the sum $(n^k + n^k + \cdots + n^k)$ is greater than the sum $(1^k + 2^k + \cdots + n^k)$ and is clearly $O(n^{k+1})$ since the sum is exactly n^{k+1}. Therefore the smaller sum of $(1^k + 2^k + \cdots + n^k)$ is also $O(n^{k+1})$.

Question 3

There are some statements that we can say about the big-O estimates involved in the problem:

1. Logarithmic functions grow slower than all positive powers of n (i.e. $\sqrt{n}$, n, n^2, n^3, etc.)
2. Exponential functions grow faster than polynomial functions
3. Factorials grow faster than exponential functions

With these statements in mind, we can order the given functions as such:

$$(\log n)^3, \sqrt{n}\log n, n^{99} + n^{98}, n^{100}, (1.5)^n, 10^n, (n!)^2$$

We use Statement 1 to say that $(\log n)^3$ is the slowest growing function. The next three can be placed in order of power of n due to Statement 2 ($\frac{1}{2}$, 99, and 100, respectively). Statement 2 also puts the two exponential functions afterwards in order of base. Finally, the statement $(n!)^2$ is the fastest growing function due to Statement 3.

Question 4

The C code block from page 50 and Exercise 2-9 of the C Programming Language by Kernighan and Ritchie, the second edition.

```
int bitCount(unsigned x) {
    int count;

    for (count = 0; x != 0; x &= (x - 1))
        count++;
    return count;
}
```

(a) The best way to show that the above function returns the number of 1 bits in the unsigned integer `x` is to use an example of the computation of unsigned int `x = 7`.

The first iteration of the loop would compare `x = 7` and `x = 6`.

```
  0111
& 0110
------
  0110
```

The second iteration has a `count = 1` with `x = 6` and it would compare `x = 6` and `x = 5`.

```
  0110
& 0101
------
  0100
```

The third iteration has a `count = 2` with `x = 4` and it would compare `x = 4` and `x = 3`.

```
  0100
& 0011
------
  0000
```

Now the `count = 3` and `x = 0` so the loop would end. The `count` now equals the number of 1 bits in the unsigned integer 7 (`0111`).

(b) The number of iterations equals the number of `count` which equals the number of 1 bits in the unsigned integer `x`.

Question 5

We can find which method is more efficient by analyzing the matrix multiplication operations for each method. In the case of `(AB)C`, you calculate the number of operations in `AB` and add it to the operations of multiplying `C` and the resulting matrix of `AB`. Similarly for `A(BC)`, you calculate the number of operations in `BC` and add it to the operations of multiplying `A` and the resulting matrix of `BC`.

For the matrices A, B, and C with dimensions 3×9, 9×4, and 4×2, respectively:

$$\texttt{(AB)C} = (3 \times 9 \times 4) + (3 \times 4 \times 2) = 132 \text{ integer multiplications}$$

$$\texttt{A(BC)} = (9 \times 4 \times 2) + (3 \times 9 \times 2) = 126 \text{ integer multiplications}$$

The above calculations show that `A(BC)` is more efficient than `(AB)C` while maintaining the resulting 3×2 matrix.