Homework 1

Date: 2023-06-16 13:31
Type: #Homework
Class: CS 311 - Algorithm Analysis & Design

Big-O & Runtime Analysis

Question 1

If you are showing that if $f \in O(g)$, then you must show that there is a constant $c \geq 1$ such that for every $n \geq 1$, $f(n) \leq cg(n)$. Your proof must state the value of the constant c, and, if your inequality only holds for values of n that are above some threshold N, then you must provide the value of N as well.

If you are showing that $f \notin O(g)$, then you must show that for every constant $c \geq 1$ and every threshold $N > 0$, there is always an $n > N$ for which $f(n) \geq cg(n)$.

You may use the fact that for every $k > 0$ and $a > 0$, $n^k \in O(n^{k+a})$, and $\log^k n \in O(n^a)$.

1. Is $8n^3 + 9n^2 + 5 \in O(n^3)$?

$n^3 \leq 8n^3 + 9n^3 + 5n^3$
$n^3 \leq 23n^3$
$f(n) \leq c * g(n), \text{where } c = 23, \text{ therefore, } 8n^3 + 9n^2 + 5 \in O(n^3)$

2. is $2^{2^{n+2}} \in O(2^{2^{n+1}})$?

$$\lim_{n\to\infty} \underbrace{\frac{2^{2^{n+1}}}{2^{2^{n+2}}}}_{\text{divide terms}} = \lim_{n\to\infty} \underbrace{\frac{1}{2^{2^{n+1}}}}_{\text{goes to } \infty} = 0$$

$\text{because } \lim_{n\to\infty} \frac{f(n)}{g(n)} = 0 \text{ , } 2^{2^{n+2}} \in O(2^{2^{n+1}})$

3. Prove that if $f \in O(g)$ and h is any positive-valued function, then $fh \in O(gh)$.

if $f \in O(g)$, then $f(n) \leq c * g(n)$. If h is any positive-valued function, meaning nonnegative, then $h > 0$ and $f(n) * h(n) \leq c * g(n) * h(n)$. because we are multiplying by the same thing on both sides & the h function is positive (meaning it will never inverse anything), then the inequality would still hold true for any non-negative function h because $g(n)$ is already $\leq$ to $f(n)$. (synonamous with multiplying both sides by 1). Therefore $fh \in O(gh)$.

Question 2

Formally derive the runtime of each algorithm below as a function of n and determine its Big-O upper bound.

a)

```java
int r = 0;
for (int i=1; i<=n;i++) {
        for (int j=i; j<=n;j++){
                for (int k=1; k<=j-1; k++){
                        r++;
                }
        }
        print(r);
}
```

$$T(n) = \underbrace{\sum_{i=1}^{n}}_{\text{outer loop}} \underbrace{\sum_{j=i}^{n}}_{\text{2nd loop}} \underbrace{\sum_{k=1}^{j-i}}_{\text{inner loop}} 1 = \sum_{i=1}^{n}\sum_{j=i}^{n}(j-i) \rightarrow \sum_{i=1}^{n}\sum_{j=i}^{n} \underbrace{l}_{\text{substitution}}$$

$$\underbrace{\sum_{i=1}^{n} \frac{(n-i)(n-i+1)}{2}}_{\text{sum of natural numbers}} = \sum_{i=1}^{n} \frac{n^2-n-i(2n-1)-i^2}{2} U = \underbrace{\frac{n^3-n^2-\frac{n(n+1)}{2}(2n-1)-\frac{n(n+1)(2n+1)}{6}}{2}}_{\text{sum of natural numbers}}$$

$$= \frac{n(n^2-1)}{6} = \frac{n^3}{6} - \frac{n}{6} = O(n^3)$$

b)

```java
int x = Math.pow(2, n);
for (int i=1; i<=x; i=i*2){
        for (int j=1; j<=i; j++){
                print("hello");
        }
}
```

$x = 2^n$, The for loop run in $\log_2 x$

$$T(n) = 1+2+4+8+\cdots+2^n \rightarrow \underbrace{\frac{1(2^{n+1}-1)}{2-1}}_{\text{sum of numbers}} = 2^{n+1}-1 = O(2^n)$$

c)

```java
int i = n;
while (i >= 1){
        for (int j=1; j<=i; j++){
```

```
                print("hello");
            }
            i = i/2;
}
```

$$T(n) = \underbrace{n + \frac{n}{2} + \frac{n}{4} + \frac{n}{8} + \cdots + 2 + 1}_{i=\frac{i}{2}} = n\left(1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \ldots\right)$$

$$= \underbrace{\frac{1}{1-\frac{1}{2}}}_{\text{infinite sum}} = 2n = O(n)$$

Question 3

Given an array of integers $A = [a_0, a_1, a_2, \ldots, a_{n-1}]$, let p_A be the following polynomial

$$a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \cdots + a_2x^2 + a_1x + a_0$$

Give an algorithm that gets an array A and an integer t as inputs and outputs $p_A(t)$ (value of the polynomial at t). Derive the worst-case run-time of your algorithm (as a function of the size of the input array). Your grade depends partly on the runtime of the algorithm. Remember that `Math.pow(x,y)` is not a primitive operation. Write in clear pseudocode.

using horner's method:

```
public foo(arr[] A, int t){
        int eval = 0; //current evaluation
        for (int i=A.length-1; i >= 0; i--){
                eval = eval*t + A[i];
        }
        return eval;
}
```

the algorithm has 1 for loop that depend on the size of the array and does arithmetic for each iteration, so it does constant steps each pass through, so:

$$T(n) = \underbrace{\sum_{i=n}^{0}}_{\text{flipped } 0 \to n} 1 \implies O(n)$$

Question 4

Write an algorithm that gets an array of integers A as input and computes the median. Derive the run time of your algorithm. Express the runtime as a function n, the size of the array.

```
public foo(arr[] A){
        for (int i=1; i < A.length-1; i++){
                temp = A[i]
                int j = i-1;
                while (j > -1 && A[j] > temp) {
                        A[j+1] = A[j]
                        --j;
                }
                A[j+1] = temp;
        }
        if (A.length%2==0){
                return (A[A.length/2] + A[(A.length/2)+1])/2
        }
        else
                return A[A.length/2]
}
```

This algorithm sorts the array via Insertion sort and then returns the value in the middle of the list. Because it depends on the insertion sort algorithm, it runs in a worst case runtime of $O(n^2)$, but typically runs in $O(n)$ for nearly sorted lists.