Algorithmic Analysis

Discussion 07

Announcements

- HW 4 due Tuesday 03/01
- Enigma due Friday 03/04

Review

Cost

Time Complexity (Time Cost) - How long does it take to run this program if we feed it certain input?

Space Complexity (Spatial Cost) - How much space does this program take to run on our computer?

Asymptotics

Asymptotics allow us to evaluate the performance of our programs using math. We ignore all constants and only care about the value with reference to the input (usually defined as N)

Big O - The upper bound in terms of the input (essentially, assume every conditional statement evaluates to the worst case).

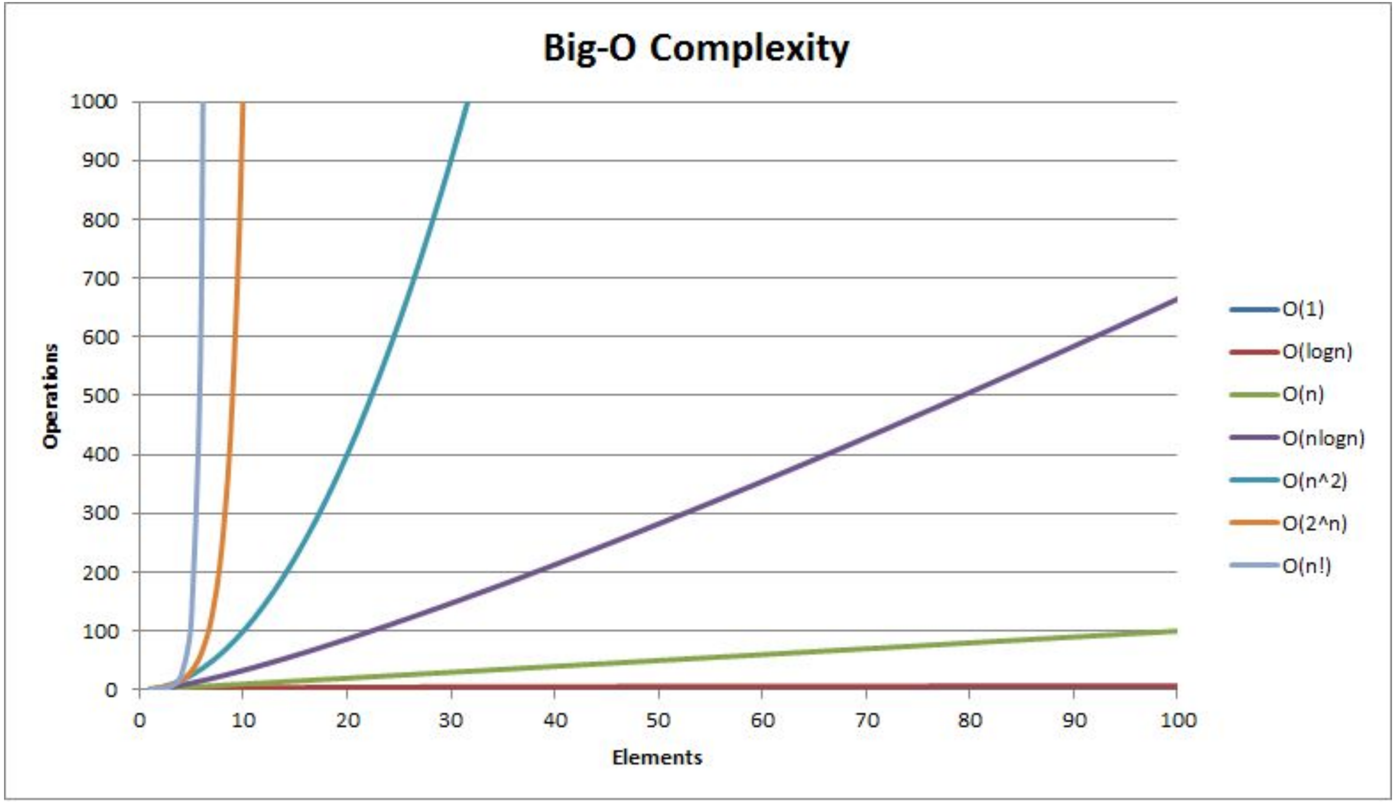
Big Ω - The lower bound in terms of the input (essentially, assume every conditional statement evaluates to the best case).

Big Θ - The tightest bound, which only exists when the tightest upper bound and the tightest lower bound converge to the same value.

Fun sums:

$$1 + 2 + 3 + \ldots + N = \Theta(N^2)$$

$$1 + 2 + 4 + \ldots + N = \Theta(N)$$


Big-O Complexity
1000
900
800
700
600
500
400
300
200
100
0
Operations
0
10
20
30
40
50
60
70
80
90
100
Elements
O(1)
O(logn)
O(n)
O(nlogn)
O(n^2)
O(2^n)
O(n!)

CS 61B Spring 2022 — Algorithmic Analysis

Exam Prep Discussion 7: February 28, 2022

1 Asymptotics Introduction

Give the runtime of the following functions in Θ notation. Your answer should be as simple as possible with no unnecessary leading constants or lower order terms.

NOTE: I computed exact runtimes using math formulas; this is not necessary to find general order of growth i.e. N^2, but is included for those that are interested.

```
private void f1(int N) {
    for (int i = 1; i < N; i++) {
        for (int j = 1; j < i; j++) {
            System.out.println("hello tony");
        }
    }
}
```

Series:

Iteration	1	2	3	...	n
Work	1	2	3	...	n

Total Work: $1+2+3+\ldots+n = \frac{(n+1)n}{2} = \Theta(N^2)$

Arithmetic Series

$\Theta(\underline{N^2})$

```
private void f2(int N) {
    for (int i = 1; i < N; i *= 2) {
        for (int j = 1; j < i; j++) {
            System.out.println("hello hannah");
        }
    }
}
```

Total Work: $1+2+4+\ldots+N = \frac{1(1-2^{\log_2 N})}{1-2} = 2^{\log_2 N} - 1 = N - 1 = \Theta(N)$

Geometric Series (Dominating Sum)

$\Theta(\underline{N})$

2 Finish the Runtimes

Below we see the standard nested for loop, but with missing pieces!

```
for (int i = 1; i < ______; i = ______) {
    for (int j = 1; j < ______; j = ______) {
        System.out.println("We will miss you next semester Akshit :(");
    }
}
```

For each part below, **some** of the blanks will be filled in, and a desired runtime will be given. Fill in the remaining blanks to achieve the desired runtime! There may be more than one correct answer.

Hint: You may find `Math.pow` helpful.

(a) Desired runtime: $\Theta(N^2)$

```
for (int i = 1; i < N; i = i + 1) {
    for (int j = 1; j < i; j = j+1___) {
        System.out.println("This is one is low key hard");
    }
}
```

(b) Desired runtime: $\Theta(log(N))$

```
for (int i = 1; i < N; i = i * 2) {
    for (int j = 1; j < _2____; j = j * 2) {
        System.out.println("This is one is mid key hard");
    }
}
```

Outer loop runs in log time already so inner loop needs to be in constant time.

(c) Desired runtime: $\Theta(2^N)$

Math.pow(2,i)

```
for (int i = 1; i < N; i = i + 1) {
    for (int j = 1; j < ______; j = j + 1) {
        System.out.println("This is one is high key hard");
    }
}
```

(d) Desired runtime: $\Theta(N^3)$ Math.pow(2,N)

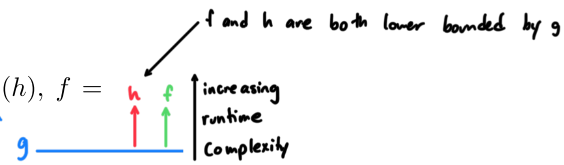
```
for (int i = 1; i < ______; i = i * 2) {
    for (int j = 1; j < N * N; j = j+1___) {
        System.out.println("yikes");
    }
}
```

Outer loop is in N time, inner loop is in N^2 time

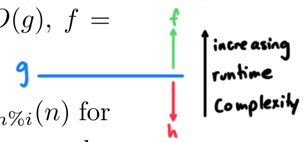
3 Asymptotic Expressions

(a) Which of the following expressions are true? Check all that apply. Equations between asymptotic expressions, such as $O(f) = O(g)$ simply mean that all functions that are $O(f)$ are also $O(g)$ and vice-versa. An expression such as $O(f) \subseteq O(g)$ means that all functions that are $O(f)$ are also $O(g)$.

- ☒ $\Theta(1000 * N^3 + N * \log(N)) = \Theta(N^3)$.
- ☒ For all $k \geq 0$, $O(N^k) \subseteq O(N^{k+1})$. i.e. $O(N) \subseteq O(N^2)$
- ☐ For all $k \geq 0$, $\Omega(N^k) \subseteq \Omega(N^{k+1})$. i.e. $\Omega(N) \not\subseteq \Omega(N^2)$
- ☐ For positive-valued functions f and g, if $f = \Omega(g)$ and $g = O(h)$, $f = \Omega(h)$. f is lower bounded by g and g is upper bounded by h - f's position relative to h is unknown.



- ☒ For positive-valued functions f and g, if $f = \Omega(g)$ and $h = O(g)$, $f = \Omega(h)$. f is lower bounded by g and h is upper bounded by g, so f is lower bounded by h necessarily (see diagram)



(b) For positive-valued functions $f_0 \ldots f_k$, where we define $f_i(n) = 1 + f_{n\%i}(n)$ for $i \geq 1$ and $f_0(n) = 1$, which of the following are true? Check all that apply. Assume that $n > k$.

- ☐ The evaluation of $f_k(n)$ may run forever. n % i is upper bounded by i-1; it will go from $f_k \to f_{k-1} \ldots \to f_0$
- ☐ $f_k(n) = \Omega(log(k))$, with respect to k. could be constant time if n is a multiple of k
- ☒ $f_k(n) = O(k)$, with respect to k. worst case scenario goes $k \to k-1 \to \ldots$ (see first part)
- ☒ $f_k(n) = \Theta(1)$, with respect to n. n is relevant to finding next function but the k bounds it
- ☒ If $n = k! - 1$, $f_k(n) = \Theta(k)$, with respect to k. k!-1 guarantees the worst case $k \to k-1 \to \ldots$ since k!-1 is not divisible by 1 to k since it is just off by 1 every time

4 Prime Factors

Determine the best and worst case runtime of `prime_factors` in $\Theta(.)$ notation as a function of `N`.

```
int prime_factors(int N) {
    int factor = 2;
    int count = 0;
    while (factor * factor <= N) {
        while (N % factor == 0) {
            System.out.println(factor);
            count += 1;
            N = N / factor;
        }
        factor += 1;
    }
    return count;
}
```

Best Case: $\Theta(\log N)$, Worst Case: $\Theta(\sqrt{N})$

when N is power of 2, factor never increments

when N is prime, factor is always incrementing