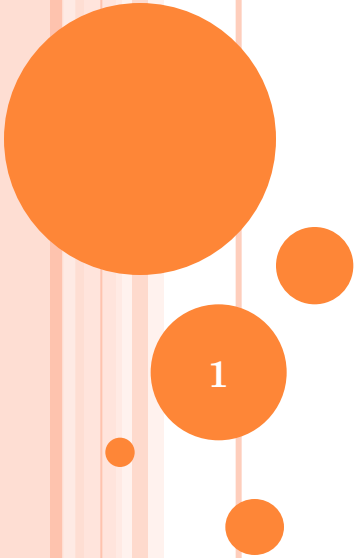


COMP 352

Tutorial 1: Algorithm Analysis

Outline

- Quick Overview on Asymptotic Analysis:
  - Definition
  - Asymptotic Notations
  - Review on Logarithm and Exponential Functions
- Exercises

Analysis and Complexity of Algorithms:

What is an Algorithm?

What do we mean by Complexity ?

How to measure Complexity ?

Analysis and Complexity of Algorithms

- Algorithm is a sequence of computational steps that transform the input into the output.

- Analysis of an algorithm is to determine the amount of resources (such as time and storage) necessary to execute it.

Analysis and Complexity of Algorithms

- The complexity of an algorithm is the cost, measured in running time, or storage, or whatever units are relevant, of using the algorithm to solve one of those problems.

Big-O

- Big-O notation is a relative representation of the complexity of an algorithm.

- Big-O is just a way to "Express" yourself in a common way, "How much time / space does it take to run the code?"

Asymptotic Notations

O-notation (upper bounds):

We write $f(n) = O(g(n))$ if there exist constants $c > 0$, $n_0 > 0$ such that $0 \le f(n) \le cg(n)$ for all $n \ge n_0$.

Example: $2n^2 = O(n^3)$ $(c = 1, n_0 = 2)$

functions, not values

THE SEVEN FUNCTIONS ALONG WITH THEIR CORRESPONDING GROWTH RATE:

T (n)	Name	Problems
O(1)	Constant	Easy to Solve
O($logn$)	Logarithmic	
O(n)	Linear	
O(n$logn$)	Linear-logarithmic	
O(n^2)	Quadratic	
O(n^3)	Cubic	
O(2^n)	Exponential	Hard to Solve
O(n!)	Factorial	

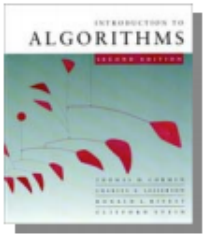
Asymptotic Notations

O-notation is an *upper-bound* notation. It makes no sense to say $f(n)$ is at least $O(n^2)$.

$\Omega(g(n)) = \{ f(n) :$ there exist constants $c > 0, n_0 > 0$ such that $0 \le cg(n) \le f(n)$ for all $n \ge n_0 \}$

Example: $\sqrt{n} = \Omega(\lg n) \quad (c = 1, n_0 = 16)$

Asymptotic Notations



Θ-notation (tight bounds)

$$\Theta(g(n)) = O(g(n)) \cap \Omega(g(n))$$

IMPORTANT REVIEWS:

- The sum of consecutive integer numbers from i to n:

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

- Properties of:

Logarithms

$\log_b(x \cdot y) = \log_b x + \log_b y$

$\log_b (x/y) = \log_b x - \log_b y$

$\log_b x^a = a \cdot \log_b x$

$\log_b a = \log_x a / \log_x b$

Exponentials

$a^{(b+c)} = a^b \cdot a^c$

$a^{b \cdot c} = (a^b)^c$

$a^b / a^c = a^{(b-c)}$

$b = a^{\log_a b}$

$b^c = a^{c * \log_a b}$

TUTORIAL 1- ANALYSIS OF ALGORITHMS

Q. 1. a

What is the Big-O of the following methods?

```
public int example1 ( int A[] ) {
    // compute sum of elements of A
    int i, sum = 0;

    for ( i=0; i < A.length; i++ ) {
        sum += A[i];
    }
    return( sum );
}
```

TUTORIAL 1- ANALYSIS OF ALGORITHMS

Q. 1. b

```
public void example2 ( int A[], int B[] ) {
    // store the prefix sums of A into B
    int i,j, sum;

    for( i=0; i < A.length; i++ ) {
        sum = 0;
        for( j=0; j <= i; j++ ) {
            sum += A[j];
        }
        B[i] = sum;
    }
}
```

TUTORIAL 1- ANALYSIS OF ALGORITHMS

Q. 1. c

```
public void example3 ( int A[], int B[] ) {
    // store the prefix sums of A into B
    int i,sum;

    sum = 0;
    for( i=0; i < A.length; i++ ) {
        sum += A[i];
        B[i] = sum;
    }
}
```

TUTORIAL 1- ANALYSIS OF ALGORITHMS

Q. 1. d

```
public void example4 ( int A[], int B[] ) {
    // store the prefix sums of prefix sums of A into B
    int i,j,k,sum;

    for( i=0; i < A.length; i++ ) {
        sum = 0;
        for( j=0; j <= i; j++ ) {
            for( k=0; k <= j; k++ ) {
                sum += A[k];
            }
        }
        B[i] = sum;
    }
}
```

Tutorial 1- Analysis of Algorithms

Q. 1. e

Can *example4(A,B)* be re-written so that it runs faster? How much faster?

TUTORIAL 1- ANALYSIS OF ALGORITHMS

Q. 2

Suppose each row of an $n \times n$ array A consists of 1's and 0's such that, in any row of A, all the 1's come before any of the 0's in that row. Assuming A is already in memory, describe a method running in $O(n)$ time (not $O(n^2)$ time) for finding the row of A that contains the most 1's.

TUTORIAL 1- ANALYSIS OF ALGORITHMS

Q. 3

Given an n-element array X, Algorithm D calls Algorithm E on each element $X[i]$. Algorithm E runs in $O(i)$ time when it is called on element $X[i]$. What is the worst-case running time of algorithm D?

TUTORIAL 1- ANALYSIS OF ALGORITHMS

Q. 4

Describe a method for finding both the minimum and maximum of n numbers using fewer than $\frac{3n}{2}$ comparisons.
(Hint: First construct a group of candidate minimum and a group of candidate maximum.)

What is the time complexity?

TUTORIAL 1- ANALYSIS OF ALGORITHMS

Q. 5

Bob built a website and gave the URL only to his n friends, which he numbered from 1 to n. He told friend number i that he/she can visit the website at most i times. Now Bob has a counter, C, Keeping track of the total number of visits to the site (but not the identities of who visits). What is the minimum value for C such that Bob should know that one of his friends has visited his/her maximum allowed number of times?

Tutorial 1- Analysis of Algorithms

Q. 6

Show that if $d(n)$ is $O(f(n))$, then $ad(n)$ is $O(f(n))$, for any constant $a>0$.

Tutorial 1- Analysis of Algorithms

Q. 7

Show that if d(n) is $O(f(n))$ and $e(n)$ is $O(g(n))$, then the product of $d(n)e(n)$ is $O(f(n)g(n))$.

Tutorial 1- Analysis of Algorithms

Q. 8

The number of operations executed by algorithm A and B is $40n^2$ and $2n^3$, respectively. Determine n_0 such that A is better than B for $n \geq n_0$.

Tutorial 1- Analysis of Algorithms

Q. 9

Show that 2^{n+1} is $O(2^n)$.

Tutorial 1- Analysis of Algorithms

Q. 10

Show that $n^2 + 2n + 1$ is $O(n^2)$.

Tutorial 1- Analysis of Algorithms

Q. 11

Prove that $\log_a(n)$ is in $O(\log_b(n))$.