CS202 - Algorithm Analysis
How to Analyze Algorithms?

Aravind Mohan

Allegheny College

March 1, 2021

S

156	141	35	94	88	61	111	77
1	**2**	**3**	**4**	**5**	**6**	**7**	**8**

Algorithm - Find Least Played Song (S)
Input - A set of play counts associated with a variety of songs inside a playlist.
Output - The least played song.

$temp \leftarrow 200$
$res \leftarrow 1$
for $i = 1$ to $|S|$ **do**
    if $S[i] < temp$ **then**
        $temp \leftarrow S[i]$
        $res \leftarrow i + 1$
    end if
end for
return res;

S

156	141	35	94	88	61	111	77
1	**2**	**3**	**4**	**5**	**6**	**7**	**8**

Algorithm - Find Least Played Song (S)
Input - A set of play counts associated with a variety of songs inside a playlist.
Output - The least played song.

$temp \leftarrow S[0]$
$res \leftarrow 1$
for $i = 1$ to $|S|$ **do**
  if $S[i] < temp$ **then**
    $temp \leftarrow S[i]$
    $res \leftarrow i + 1$
  end if
end for
return res;

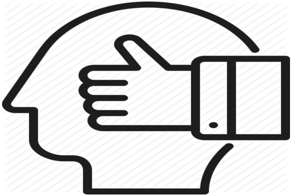
Alternative

Discussion Based On ...

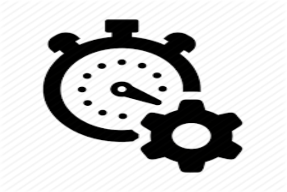
Sedgewick 1.4

The three important techniques ...

- **Algorithm:** Outline, the essence of a computational procedure, step by step instructions.
- **Program:** An implementation of an algorithm in some programming language such as Java, C, C++, Python, etc ...
- **Data structure:** Organization of data needed to solve the problem.



(a) **(Correctness)**



(b) **(Efficiency)**

How is an Algorithm different from a Program?

- Makes uses of high level description of the algorithm , instead of testing one of its implementations
- Take into account all possible inputs
- Allows one to evaluate the efficiency of any algorithm in a way that is independent of the hardware and software environment

How do we measure the running time of a Program?

- First write a program to implement the algorithm.
- Execute the program using datasets of varying size and composition.
- Integrate a Java method like
  System.currentTimeMillis()
  to get an accurate measure of the actual running time.

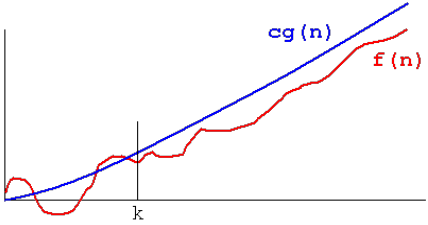
Limitation of the Experimental Study approach!

- It is necessary to implement and test the algorithm in order to determine the running time.
- Experiments can be done only on a limited set of inputs, and may not be indicative of the running time on other inputs not included in the experiment.
- In order to compare two algorithms, the same hardware and software environments should be used.

Asymptotic Analysis

- Goal: to simplify analysis of running time by getting rid of ”details”, which may be affected by specific implementation and hardware.
  - like ”rounding”: 1,000,001 = 1,000,000
  - $3n^2 = n^2$
- Capturing the essence: how the running time of an algorithm increases with the size of the input in the limit.
  - Asymptotically more efficient algorithm are best for all but small inputs.

- Mainly used for worst-case analysis.
- Referred to as Asymptotic upper bound.
- $f(n) = O(g(n))$, if there exists constants c and k , such that $f(n) \leq cg(n)$ for $n \geq k$
- $f(n)$ and $g(n)$ are functions over non negative integers



Asymptotic Growth

- Logarithmic: $O(log(n))$
- Linear: $O(n)$
- Log-Linear: $O(nlog(n))$
- Quadratic: $O(n^2)$
- Polynomial: $O(n^k)$, $k \geq 1$
- Exponential: $O(a^n)$, $a > 1$

Thumb Rule of Asymptotic Notation

- Drop lower order terms and constant factors
  - $50nlog(n)$ is $O(nlog(n))$
  - $7n - 3$ is $O(n)$
  - $8n^2log(n) + 5n^2 + n$ is $O(n^2log(n))$
- Note: Even though $(50nlog(n))$ is $O(n^5)$, it is expected that such an approximation be of as small an order as possible.

Asymptotic Analysis Example-1

Algorithm1 (n)

Initialize $i, j, sum = 0$
for $(i = 0; i < n; i = i + 1)$ **do**
    for $(j = 0; j < n; j = j + 1)$ **do**
        $sum \leftarrow i + j$
    end for
end for
return sum

O(n^2), Quadratic

Asymptotic Analysis Example-2

Algorithm2 (n)

1: Initialize $i, j, sum = 0$
2: **for** $(i = 0; i < n; i = i + 2)$ **do**
3:     **for** $(j = 0; j < n; j = j + 2)$ **do**
4:         $sum \leftarrow i + j$
5:     **end for**
6: **end for**
7: **return sum**

O(n^2), Quadratic

Asymptotic Analysis Example-3

Algorithm3 (n)

1: Initialize $x = 0, y = n$
2: **while** $(y > 0)$ **do**
3: $\quad x \leftarrow x + i$
4: $\quad y \leftarrow y/2$
5: **end while**
6: **return x**

O(log(n)), Logarithmic

Asymptotic Analysis Example-4

Algorithm4 (n)

```
Initialize i, j, sum = 0
for (i = 0; i < n; i = i + 2) do
    for (j = 0; j < n; j = j * 2) do
        sum ← i + j
    end for
end for
return sum
```

O(nlog(n)), Log-Linear

Asymptotic Analysis Example-5

Algorithm5 (n)

```
Initialize i, j, sum = 0
for (i = 1; i <= n; i = i + 1) do
    for ((j = 1; j <= i; j = j + 1) do
        while (j > 0) do
            j = j/2
            sum = i + j
        end while
    end for
end for
return sum
```

Guess?

Defective Coin Problem

Given a set of coins of size n, where $n = 2^k$
The weight of all coins are equal except one.
The one with the different weight is the **Defective**
Assume a weigh scale is given to measure the weight of coin(s) in constant time.
Write an Algorithm to detect the defective one.

Defective Coin Problem

- First, let us solve this problem. (**Brute Force**)
- Next, find a way to solve it fast.

Brute Force Algorithm

Algorithm - FDC(W)
Input - A set of coin weights associated with a collection of coins.
Output - The position of the defective coin.

```
for i = 0 to |W| - 1 do
    if W[i] <> W[i + 1] then
        if W[i] < W[i + 1] then
            return i
        else
            return i + 1
        end if
    end if
end for
```

O(n), linear time

Divide and Conquer Algorithm

Algorithm - FDC($W, low, high$)
Input - A set of coin weights associated with a collection of coins.
Output - The position of the defective coin.

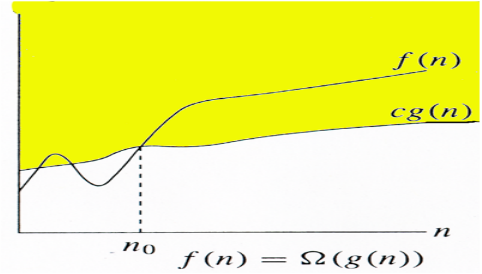
$first \leftarrow SCALE(low$ to $(low + high)/2)$;
$second \leftarrow SCALE((low + high)/2$ to $high)$;
if $(high - low)$ is equal to 1 **then**
    if $first < second$ **then**
        return low;
    else
        return $high$;
    end if
else
    if $first < second$ **then**
        return FDC($W, low, (low + high)/2$);
    else
        return FDC($W, (low + high)/2, high$);
    end if
end if

O(log(n)), logarithmic time

BIG Omega Notation

- The ”big-Omega” or Ω - notation.
- It is generally used to describe best case running time or lower bound of algorithmic problems.
- $f(n) = \Omega(g(n))$ if there exists constant c and n_0 such that $cg(n) \leq$ f(n) for $n \geq n_0$.
- E.g., lower-bound of searching in an unsorted array is $\Omega(n)$.

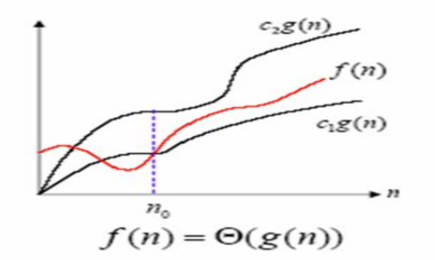
BIG Omega Notation



BIG Theta Notation

- The "big-Theta" or Θ - notation.
- Asymptotic tight bound.
- It is generally used to describe running time in between best and worst case. For example: **average** running time of an algorithmic problem.
- $f(n) = \Theta(g(n))$ if there exists constant c_1, c_2, and n_0 such that
  $c_1\ g(n) \leq$ f(n) $\leq c_2\ g(n)$ for $n \geq n_0$
- $f(n) = \Theta(g(n))$ if and only if $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.

BIG Theta Notation



Reading Assignment

Sedgewick 1.4

Questions?

Please ask if there are any Questions!