CS 3345.004 Lecture # 2

Running-time analysis of programs (RT analysis):

Given an algorithm → calculate # of operations it performs on inputs of size n.

Express RT in order notation

Interest: What happens as $n \to \infty$.

Model of ~~comp~~ RT analysis: Every operation (arithmetic, logical) takes 1 unit of time.

Order notation (Asymptotic notation):

"Big Oh": Let f, g be discrete, nonnegative functions

$f(n) = O(g(n))$ if there is a constant $c > 0$, there is some $n_0 \geq 0$. $f(n) \leq c\, g(n)$ for all $n \geq n_0$

Worst-case analysis = RT that is max for size n.

Table of feasible sizes for given RT's:

RT (O)	Size
1	-
$\log n$	$n = 10^{12..18}$ = Trillions ... Hexill
n	10^{10}

RT	Feasible size
$n \log n$	10^8
n^2	$10^{4..5}$
n^3	10^3
2^n	$< 10^2$
$n!$	< 25

Logs:

Def: $\log_b a = k \iff b^k = a$

In math/engg: default $b = e$, in CS, $b = 2$

$\log n = \log_2 n$

$\ln n = \log_e n$

Properties:

$\log(AB) = \log A + \log B$

$\log(A/B) = \log A - \log B$

$\log^k n = (\log n)^k$ ← poly log

$\log(n^k) = k \cdot \log n$

Exponentials: c^n c = constant

$x^a \cdot x^b = x^{a+b}$, $(x^a)^b = x^{ab}$

Polynomial in n: $P(n) = a_0 + a_1 n + \cdots a_k n^k$

k = constant, $a_k > 0$ $P(n) = O(n^k)$.

Thesis: Polynomial RT = feasible algorithm

Exponential or more = infeasible.

Sums of sequences:

1. Arithmetic sequences: $a_1, a_2, \dots, a_n$

$$a_{i+1} - a_i = d = \text{constant}.$$

Ex: $1, 2, 3, \dots, n$

$2, 5, 8, 11, \dots, 3n-1$

$d > 0$ = increasing sequence,
Max term = a_n

$d < 0$ = decreasing sequence
Max term = a_1

$$a_1 + a_2 + \dots + a_n = \sum_{i=1}^{n} a_i$$

$$a_n + a_{n-1} + \dots + a_1 = \sum_{i=1}^{n} a_i$$

$$(a_1 + a_n)\ (a_1 + a_n) \dots (a_1 + a_n) = 2 \sum_{i=1}^{n} a_i = n(a_1 + a_n)$$

$$\sum_{i=1}^{n} a_i = \frac{1}{2} n (a_1 + a_n) = O(n \cdot \text{biggest term})$$

2. Geometric sequences: $a_0, a_1, \dots, a_n$

$$a_{i+1} / a_i = r = \text{constant}.$$

$r > 1$ = increasing sequence
Max term = a_n

$r < 1$ = decreasing sequence
Max term = a_0

Ex: $1, 2, 4, 8, 16, \dots, 2^n$

$$\sum_{i=0}^{n} a_i = \frac{a_0 (r^{n+1} - 1)}{r - 1} = O(\text{bigg... ter...}$$

Size $n = r - p + 1$

RT analysis: Merge algorithm

Algorithm $Merge(A, p, q, r)$ // Pre: $A[p..q]$, $A[q+1..r]$ are sorted subarrays

Copy $A[p..q]$ into $L[1..q-p+1]$ — $q - p + 1$
Copy $A[q+1..r]$ into $R[1..r-q]$ — $r - q$ $\Big\}$ $r-p+1 = n$
Set sentinels $L[q-p+2] = R[r-q+1] = \infty$ - 2
$i \leftarrow 1$; $j \leftarrow 1$ — 2
for $k \leftarrow p$ **to** r **do**
  if $L[i] \le R[j]$ **then**
    $A[k] \leftarrow L[i]$; $i \leftarrow i + 1$ — 3
  else
    $A[k] \leftarrow R[j]$; $j \leftarrow j + 1$ — 3

$1 + \max(3,3) = 4$

$(r-p+1)(1+4) = 5n$

Total RT $= 6n + 4 = O(n)$.

Array $A[p..r]$ Size $n = r - p + 1$

RT analysis: Partition algorithm

$i \leftarrow Random(p, r)$ — 1
Exchange $A[i]$ and $A[r]$ - 3
$x \leftarrow A[r]$ — 1
$i \leftarrow p - 1$ — 2
for $j \leftarrow p$ **to** $r - 1$ **do**
  if $A[j] \le x$ **then**
    $i \leftarrow i + 1$ — 2
    Exchange $A[i]$ and $A[j]$ — 3
Exchange $A[i+1]$ and $A[r]$ — 3
return $i + 1$ — 2

$\max(1+5, 1+0) = 6$

$(r-p)(1+6) = (n-1)7$

Total RT $= 12 + (n-1)7 = 7n + 5 = O(n)$

Size $n = r - p + 1$

RT analysis: Binary search

```
BSearch(A, p, r, x) // Search for x in sorted array A[p..r]
while p ≤ r do
    q ← (p + r)/2
    if x < A[q] then
        r ← q − 1
    else if x > A[q] then
        p ← q + 1
    else
        return q // index of x in A
return -1 // return -1 if x is not in A
```

$O(1)$ ← Each time loop is executed, $n = r-p+1$ shrinks by $1/2$

\# of iterations of while loop: $1 + \log n$

$RT = O(\log n)$.

$n \to n/2 \to n/4 \to \cdots \to 0$

\# steps: $1 + \log n$

Size $= n$

RT analysis: Bubble sort

```
BubbleSort(A, n) // Sort A[1..n]
for i ← 1 to n do
    for j ← n downto i + 1 do
        if A[j] < A[j − 1] then
            Exchange A[j] and A[j − 1]
```

$O(1)$

$O(n-i)$

$$\sum_{i=1}^{n} n - i = (n-1) + (n-2) + \cdots + 1 \quad \leftarrow \text{decreasing arithmetic sequence}$$

$$= O(n^2)$$

S

Amortized analysis

RT analysis: KMP algorithm for string matching

A useful loop invariant: $\pi[k] < k$, for $k > 0$.

```
π[1] ← 0
k ← 0
for q ← 2 to m do
    while k > 0 and P[k+1] ≠ P[q] do
        k ← π[k]
    if P[k+1] = P[q] then
        k ← k + 1
    π[q] ← k
return π
```

← Total RT of while loop $= O(\#\text{ of down steps})$

$\#$ of down steps $\leq$ $\#$ of up steps $= m-1$

$O(m)$

Asymptotic notation

Let f and g be discrete, non-negative functions. In the following definitions, b, c and n_0 are positive constants.

- Upper bound (big oh): $f(n) = O(g(n))$, if there exist c and n_0 such that $f(n) \leq c \cdot g(n)$ for $n \geq n_0$.
- Lower bound (big omega): $f(n) = \Omega(g(n))$, if there exist c and n_0 such that $f(n) \geq c \cdot g(n)$ for $n \geq n_0$.
- Tight bound (theta): $f(n) = \Theta(g(n))$, if there exist b, c, and, n_0 such that $b \cdot g(n) \leq f(n) \leq c \cdot g(n)$ for $n \geq n_0$.
- Weak upper bound (little oh): $f(n) = o(g(n))$, if for any c, there exists n_0 such that $f(n) < c \cdot g(n)$ for $n \geq n_0$.
- Weak lower bound (little omega): $f(n) = \omega(g(n))$, if for any c, there exists n_0 such that $f(n) > c \cdot g(n)$ for $n \geq n_0$.

The limit method

The following rules can be derived from the definitions, and are more convenient to derive asymptotic bounds, when the limit $k = \lim_{n\to\infty} \frac{f(n)}{g(n)}$ exists. Often, L'Hôpital's rule is used to calculate this limit.

- $f(n) = O(g(n))$, if $k \neq \infty$.
- $f(n) = \Omega(g(n))$, if $k \neq 0$.
- $f(n) = \Theta(g(n))$, if $0 < k \neq \infty$.
- $f(n) = o(g(n))$, if $k = 0$.
- $f(n) = \omega(g(n))$, if $k = \infty$.

Proof techniques

The following are valid techniques for writing proofs:

- Induction. Prove that $T(n) = 2^n - 1$ is the solution to the recurrence $T(n) = 2T(n-1) + 1, T(1) = 1$. Prove that $\lim_{n\to\infty} \frac{\log^k n}{n^\epsilon} = 0$, for any positive constants k and ϵ.
- Contradiction. If f is a maximum flow, then the residual network G_f does not have a path from s to t.
- Contrapositive. If the product of two integers is even, then at least one of them must be even.
- Example/Counterexample. If $S, V - S$ is a cut crossed by an edge e of a minimum spanning tree T, prove that it is not necessarily a light edge for this cut.
- Construction. If f is a maximum flow, then there exists a cut (S, T) such that $|f| = c(S, T)$.

Proof by induction

Given a sequence of propositions, $P_0, P_1, P_2, \ldots$, we can show that they are true as follows:

- **Base cases:** From first principles, prove that $P_0, P_1, P_2, \ldots, P_{n_0}$ are true, for some $n_0 \geq 0$.
- **Induction step:** Prove that for any $n > n_0$, $P_0, \ldots, P_{n-1}$, the Induction hypothesis (IH), implies P_n:

$$P_0 \wedge \ldots \wedge P_{n-1} \implies P_n$$

- Based on the above, we can claim P_n is true, for $n \geq 0$. Why? Suppose $k \geq 0$ is smallest value for which P_k is false. By the base case, $k > n_0$. Since k is smallest counterexample, $P_0, P_1, P_2, \ldots, P_{k-1}$ are true. By the induction step, P_k is true, which contradicts the claim that P_k is false.

Loop invariants

Proving correctness of programs with loops:

// Precondition: Pre
while Condition c **do**
  Statements S
// Postcondition: $Post$ (to be proved)

To prove that the postcondition is satisfied, we write a mathematical proposition, known as a Loop Invariant (LI), and a proof structured as follows:

- Initialization: Pre implies LI
- Maintenance: LI, c, and execution of S, implies LI.
- Termination: LI, and, not c, implies $Post$.

Show that the number of iterations is finite separately, or include it as part of the LI.