Unit-1 DAA

7th February, 2024 1

Algo. Design and Analysis Problem

Understand the Problem

↓

Decide on computational means
Exact vs approx. solution
Data Structures

→ Algo Design Technique ←

↓

→ Design an Algo. ←

↓

Prove correctness

↓

Analyze the Algo

↓

Code the Algo

→ Specifying an algo
- Natural Language
- Pseudo code
- Flowchart

→ Analyzing an algo
- Efficiency
  - Time
  - Space
- Simplicity
- Generality

→ Coding an algo
- Efficient implementation
- Correctness of program
- Code optimization

Analysis Framework

→ Complexity of an algo.

Algo's Performance

- internal factors
  - Time
  - Space
- external factors
  - speed of PC
  - quality of compiler
  - size of ip algo.

→ Space Complexity

$$S(P) = C + SP(I)$$

C: Fixed space requirements (C) independent of the characteristics of ips and ops

$SP(I)$: Variable Space Requirements (SP(I)) dependent on instance characteristic I

→ Time Complexity
- Experimental Study
- Theoretical Analysis

$$T(P) = C + T_p(I)$$

C: Compile Time (C) independent of instance characteristics

$T_p(I)$: run (execution) time TP

→ Theoretical Analysis
1. Order of magnitude / Asymptotic categorization
2. Estimation of running Time.
   1. Operation counts
   2. Step counts

Basic operation count $T(n) \approx c_{op} C(n)$

$T(n)$: Running Time

c_{op}: execution time for basic operation

$C(n)$: no. of times basic operation is executed

Asymptotic Notations

$O(g(n))$: class of functions $f(n)$ that grow no faster than $g(n)$ upper bound

$\Omega(g(n))$: atleast as fast as $g(n)$ - lower bound

$\Theta(g(n))$: at same rate as $g(n)$ - average bound

$o(g(n))$: slower rate than $g(n)$

$\omega(g(n))$: faster rate than $g(n)$

class : $1 < \log n < \sqrt{n} < n < n\log n < n^2 < n^3 < \dots < 2^n < 3^n < \dots < n^n$

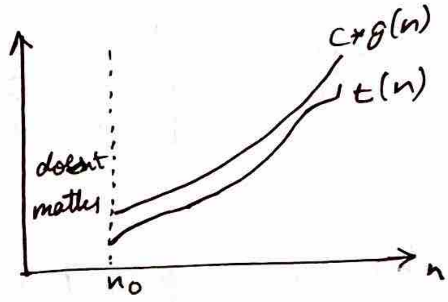
Lower bound

avg bound

upper bound

O - notation

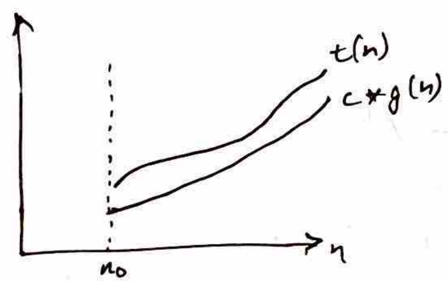
$t(n) \le c * g(n)$ for all $n \ge n_0$



Ω - notation

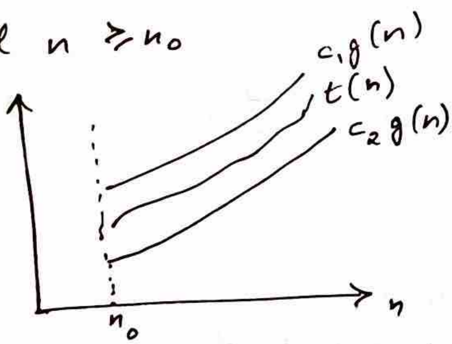
$t(n) \ge c * g(n)$ for all $n \ge n_0$

$t(n) \in O(g(n))$



Θ - notation

$c_2 g(n) \le t(n) \le c_1 g(n)$ for all $n \ge n_0$



Little-o notation

$0 \le t(n) < c * g(n)$ for all $n \ge n_0$

Little Omega notation

$t(n) > c * g(n) \ge 0$ for all $n \ge n_0$

Theorems

1. $t_1(n) \in O(g_1(n))$, $t_2(n) \in O(g_2(n))$
then $t_1(n) + t_2(n) \in O(\max\{g_1(n), g_2(n)\})$

2. $t_1(n) \in \Theta(g_1(n))$, $t_2(n) \in \Theta(g_2(n))$
then $t_1(n) + t_2(n) \in \Theta(\max\{g_1(n), g_2(n)\})$

3. $t_1(n) \in \Omega(g_1(n))$, $t_2(n) \in \Omega(g(n))$
then $t_1(n) + t_2(n) \in \Omega(\max\{g_1(n), g_2(n)\})$

Using limits to compare order of growth

$$\lim_{n\to\infty} \frac{t(n)}{g(n)} = \begin{cases} 0 & \Rightarrow t(n) \text{ has a smaller order of growth than } g(n) \\ c & \Rightarrow t(n) \quad \text{same} \quad g(n) \\ \infty & \Rightarrow t(n) \quad \text{greater} \quad g(n) \end{cases}$$

8th February, 2024

Time Efficiency of Non-recursive Algos.

eg1: Max element

```
maxvalue ← A[0]
for i ← 1 to n-1 do
        if A[i] > maxvalue
                maxvalue ← A[i]
return maxvalue
```

$$C(n) = \sum_{i=1}^{n-1} 1 = (n-1-1+1) = n-1 \in \Theta(n)$$

0 - 5

1 2 3 4 5
0 1 2 3 4
i j j j

eg2: Unique Elements

```
for i ← 0 to n-2 do
    for j ← i+1 to n-1 do
        if A[i] == A[j] return false
```

$$C_{worst}(n) = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1 = \sum_{i=0}^{n-2} ((n-1)-(i+1)+1)$$

$$= \sum_{i=0}^{n-2} (n-1-i)$$

$$= \sum_{i=0}^{n-2} (n-1) - \sum_{i=0}^{n-2} i =$$

$$(n-1)\sum_{i=0}^{n-2}(1) - \sum_{i=0}^{n-2} i$$

$$= (n-1)(n-2-0+1) - \frac{(n-2)(n-1)}{2}$$

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

$$= (n-1)\left[(n-1) - \frac{(n-2)}{2}\right]$$

$$= (n-1)\left[\frac{2n - \cancel{2} - n + \cancel{2}}{2}\right]$$

$$= \frac{(n-1)(n)}{2} \approx \frac{1}{2}n^2$$

Best - case : 1 comparision
Worst - case : $\frac{n^2}{2}$ comparisions

$$T(n)_{\text{worst case}} = O(n^2)$$

eg 3 Matrix Multiplication

```
for i ← 0 to n-1 do
    for j ← 0 to n-1 do
        C[i, j] ← 0.0
        for k ← 0 to n-1 do
            C[i, j] ← C[i, j] + A[i, k] * B[k, j]

retur[redacted] C
```

$$M(n) \in \Theta(n^3)$$

Mathematical analysis of Recursive Algo.

→ Important Recurrence Types

1. Decrease-by-one recurrences < linear / quadratic

   eg: n!

   $$T(n) = T(n-1) + f(n)$$

2. Decrease-by-a-constant factor recurrences

   eg: Binary search

   $$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

→ Methods to solve recurrences

- Substitution Method
  - Mathematical induction
  - Backward substitution
- Recursion Tree Method
- Master Method (decrease by constant factor recurrences)

eg: n!

```
F(n)
if n == 0   return 1
else return F(n-1) * n
```

Overall Time complexity : $\Theta(n)$

$n = 4$

2

1

eg: counting no. of binary digits

```
BinRec (n)
   if n == 1   return 1
   else return BinRec(⌊n/2⌋) + 1
```

eg. Tower of Hanoi

$$\Theta(2^n)$$

0	1	2	3	4	5	6
89	45	68	90	29	34	17

$n = 7$

Brute Force

- Selection sort
- Bubble sort

→ Selection sort

algo: for $i \leftarrow 0$ to $n-2$ do
  min $\leftarrow i$
  for $j \leftarrow i+1$ to $n-1$ do
    if $A[j] < A[min]$
      min $\leftarrow j$
  swap $A[i]$ and $A[min]$

$$C(n) = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1 = \sum_{i=0}^{n-2} (n-1-(i+1)+1) = \sum_{i=0}^{n-2} (n-1-i)$$

$$= (n-1) + (n-2) + \dots + \underbrace{(n-1-n+2)}_{=1}$$

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

$$\lim_{n\to\infty} \frac{n^2-n}{2(n^2)} = \cancel{\lim_{n\to\infty} \frac{2n-1}{}} = \lim_{n\to\infty} \frac{1}{2} - \frac{1}{2n} = \frac{1}{2}$$

$$\Theta(n^2)$$

→ Bubble sort

algo: for $i \leftarrow 0$ to $n-2$ do
  for $j \leftarrow 0$ to $n-2-i$ do
    if $A[j+1] < A[j]$
      swap $A[j]$ and $A[j+1]$

$$C(n) = \sum_{i=0}^{n-2} \sum_{j=0}^{n-2-i} 1 = \sum_{i=0}^{n-2} [n-2-i-0+1] = \sum_{i=0}^{n-2} (n-1-i)$$

$$= \frac{(n-1)n}{2} \Rightarrow \Theta(n^2)$$

→ Sequential search

Algo: sequentialsearch (A[0..n], k) {

```
A[n] ← k
i ← 0
while A[i] ≠ k do
    i ← i+1
if i < n return i
else return -1
```

$\theta(n)$

→ String Matching

Algo: BruteforceStringMatch (T[0..n-1], P[0..m-1])

```
for i ← 0 to n-m do
    j ← 0
    while j < m and P[j] = T[i+j] do
        j ← j+1
    if j == m return i
return -1.
```

$O(nm)$

Unit - 2 DAA

Decrease And Conquer

(Inductive / Incremental approach)

- Decrease by a constant (usually by 1)
  - insertion sort
  - graph traversal (DFS, BFS)
  - topological sorting
  - algos for generating permutations & subsets
- Decrease by a constant factor (usually by half)
  - binary search & bisection method
  - Exponentiation by squaring
  - Multiplication à la russe
- Variable - size decrease
  - Euclid's algo
  - Selection by partition
  - Nim - like games

→ Insertion Sort

eg1: 6, 4, 1, 8, 5

```
6 | 4 1 8 5
4 6 | 1 8 5
1 4 6 | 8 5
1 4 6 8 | 5
1 4 5 6 8
```

eg2: 89, 45, 68, 90, 29, 34, 17

```
89 | 45 68 90 29 34 17
45 89 | 68 90 29 34 17
45 68 89 | 90 29 34 17
45 68 89 90 | 29 34 17
29 45 68 89 90 | 34 17
29 34 45 68 89 90 | 17
17 29 34 45 68 89 90
```

Algo: InsertionSort (A[0..n-1])

```
//ip: A[0..n-1] of n orderable elements
//op: A[0..n-1] sorted in non decreasing order
for i ← 1 to n-1 do
        v ← A[i]                         4
   (0)  j ← i-1  (0)
                                         6 > 4
    while ( j ≥ 0 and A[j] > v) do
            A[j+1] ← A[j]                6 6 1 8 5
               j ← j-1   (-1)
    A[j+1] = v      4 6 1 8 5
```

→ Time Efficiency

$C_{worst}(n) = \frac{n(n-1)}{2} \in \Theta(n^2)$

$C_{avg}(n) \approx \frac{n^2}{4} \in \Theta(n^2)$

$C_{best}(n) = n-1 \in \Theta(n)$

→ Space efficiency : in-place

→ Generating Permutations

Algo: Johnson Trotter(n)

// ip: A positive integer n
// op: A list of all permutations of {1, ... n}

initialize the first permutation with $\overleftarrow{1}\,\overleftarrow{2} \ldots \overleftarrow{n}$
while the last permutation has mobile element do
    find its largest mobile element k
    swap k with the adjacent element k's arrow points to
    reverse the direction of all elements that are larger than k
    add the new permutation to the list

* if a component k points to a smaller number adjacent to it, then k is mobile component.

$\overleftarrow{1}\ \overleftarrow{2}\ \overleftarrow{3}\ \overleftarrow{4}$

$\overleftarrow{1}\ \overleftarrow{2}\ \overleftarrow{4}\ \overleftarrow{3}$

$\overleftarrow{1}\ \overleftarrow{4}\ \overleftarrow{2}\ \overleftarrow{3}$

$\overleftarrow{4}\ \overleftarrow{1}\ \overleftarrow{2}\ \overleftarrow{3}$

$\overrightarrow{4}\ \overleftarrow{1}\ \overleftarrow{3}\ \overleftarrow{2}$

$\overleftarrow{1}\ \overrightarrow{4}\ \overleftarrow{3}\ \overleftarrow{2}$

$\overleftarrow{1}\ \overleftarrow{3}\ \overrightarrow{4}\ \overleftarrow{2}$

$\overleftarrow{1}\ \overleftarrow{3}\ \overleftarrow{2}\ \overrightarrow{4}$

$\overleftarrow{3}\ \overleftarrow{1}\ \overleftarrow{2}\ \overleftarrow{4}$

$\overleftarrow{3}\ \overleftarrow{1}\ \overleftarrow{4}\ \overleftarrow{2}$

$\overleftarrow{3}\ \overleftarrow{4}\ \overleftarrow{1}\ \overleftarrow{2}$

$\overleftarrow{4}\ \overleftarrow{3}\ \overleftarrow{1}\ \overleftarrow{2}$

$\overrightarrow{4}\ \overrightarrow{3}\ \overleftarrow{2}\ \overleftarrow{1}$

→ Fake-Coin Problem

$W(n) = W(\lfloor n/2 \rfloor) + 1$ for $n > 1$, $W(1) = 0$

$W(n) = \log_2 n$

→ Russian Peasant Multiplication

n & m → +ve I

if n (even) ⇒ $n \cdot m = (n/2) \cdot 2m$

if n (odd) ⇒ $n \cdot m = ((n-1)/2) \cdot 2m + m$

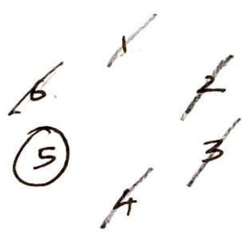
$50 \times 65 = 3250$

eg:

n	m
50	65
25	130
12	260 (+130)
6	520
3	1040
$\cancel{1}$	2080 (+1040)

ans = 2080 + 1040 + 130 = 3250

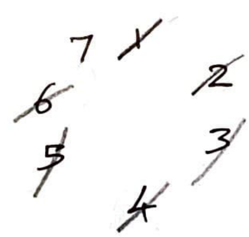
→ Josephus Problem



$J(6) = 5$

↓

1 10 ⇒ 101 = 5



$J(7) = 7$

↓

111 ⇒ 111