Name : Shubham Dutta

Year : 2nd

Steam : CST

Enrollment : 12019009022112

Registration : 301201900900752

Paper Name :- Design & Analysis of Algorithm

Paper Code : PCCCS402

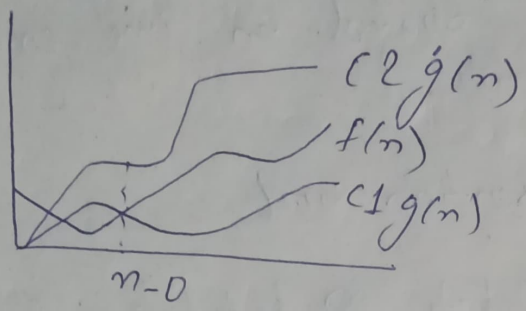
Signature : Shubham Dutta

Date : 11th March 2021

Answers

1. A) i) Asymptotic Notations are mathematical tools to represent the time complexity of algorithms for asymptotic analysis.

⇒

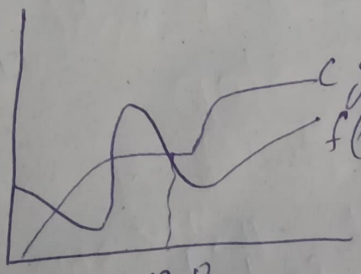


$f(n) = \Theta(g(n))$

Θ Notation :- It bounds a f= below & above, so it defines exact asymptotic behaviours.

$\Theta(g(n)) = \{f(n)$; there exist +ve constants $c1, c2$ & $n0$ such that $0 <= c1 * g(n) <= f(n) <= c2 * g(n)$ $\forall\ n >= n0$

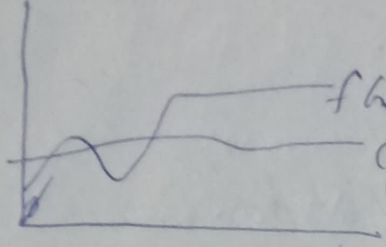
⇒



$f(n) = O(g(n))$

Big-O Notation :- It defines upper bound of algorithm, it bounds a f= only from above.
It takes worst case time complexity.

=>



$f(n) = \Omega(g(n))$

Ω Notation :- Ω Notation provides lower bound on time complexity of algorithm.

Eg :-

Insertion Sort

For Θ notation.

Time complexity =
- Worst case - $\Theta(n^2)$
- But case - $\Theta(n)$

For Big-O Notation

Time complexity = $O(n^2)$ (worst case)

For Ω Notation

Time complexity = $O(n)$ (best case)

1A) ii) Best Case :- fastest time to complete, with optimal inputs chosen.

Eg :- best case of a sorting algorithm would be data that already sorted.

•) Worst Case :- slowest time to complete with personal inputs chosen.
Eg :- worst case for a sorting algorithm might be data that's sorted in reverse order.
(basically it depends on particular algorithm)

•) Average Case :- Arithmetic mean. Run the algorithm many times using many diff. i/p of size n that come from some distribution that generates these i/p (i.e. all possible i/p are equally likely) compute the total running time & divide by the no. of trials. To normalize the result based on size of i/p sets

---

2:B) Time complexity of Insertion Sort is $O(n^2)$

A) Algorithm

Step-1 Start

Step-2 Iterate from arr[1] to arr[n] over the array

Step-3 Compare the current element (key) to its predecessor.

Step-4 If the key element is smaller than its predecessor, compare it to the elements before. Move the greater elements one position up to make space for swapped element.

Step-5 Repeat previous steps

Step-6 End.

---

3. A) Quick sort Algorithm

Step-1 Consider the 1st element of the list as pivot (i.e. element at 1st position in the list)

Step 2 Determine 2 variables i & j. set i & j to 1st & last element of list respectively.

Step 3 Increment i until list[i] > pivot then stop

Step 4 Decrement j until list[i] < pivot then stop.

Step-5 If i < j then exchange list[i] & list[j]

Step-6 Repeat steps 3, 4 & 5 until i < j

Step-7 Exchange the pivot element with list[j] element.

Worst Case

- It occurs when partition process always picks greatest or smallest element as pivot.
- Worst case occur when array is already sorted in increasing or decreasing order

$$T(n) = T(0) + T(n-1) + \Theta(n)$$

$$\Downarrow$$

$$T(n) = T(n-1) + \Theta n$$

Best Case

- It occurs when partition process always picks middle element as pivot.

$$T(n) = 2T(n/2) + \Theta(n)$$

---

4. Average Case Complexity of Quick Sort

For any pivot position i, $i \in \{0, \ldots, n-1\}$

- Time for partitioning an array, cn
- The head & tail subarrays contain i & $n-1-i$ items

$$T(n) = cn + T(i) + T(n-1-i)$$

Average running time for sorting

$$T(n) = \frac{1}{n} \sum_{i=0}^{n-1} \left(T(i) + T(n-1-i) + cn \right)$$

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

$+cn$

$nT(n) = 2(T(0) + T(1) + \dots + T(n-2) + T(n-1)) + cn^2$

$\Rightarrow (n-1)T(n-1) = 2(T(0) + T(1) + \dots T(n-2)) + c(n-1)^2$

(-)

$\Rightarrow nT(n) - (n-1)T(n-1) = 2T(n-1) + 2cn - c$

$\therefore nT(n) \simeq (n+1)T(n-1) + 2nc$

$\Rightarrow \frac{T(n)}{n+1} = \frac{T(n-1)}{n} + \frac{2c}{n+1}$

$\Rightarrow \frac{T(n)}{n+1} - \frac{T(n-1)}{n} = \frac{2c}{n+1}$

Explicit Form :-

$\Rightarrow \frac{T(n)}{n+1} + \frac{T(n-1)}{n} + \frac{T(n-2)}{n-1} + \dots + \frac{T(3)}{3}$

$+ \frac{T(n-1)}{n} - \frac{T(n-2)}{n-1} - \dots - \frac{T(1)}{2} - \frac{T(2)}{1}$

$= \frac{2c}{n+1} + \frac{2c}{n} + \dots + \frac{2c}{3} + \frac{2c}{2}$

$\Rightarrow \frac{T(n)}{n+1} = \frac{T(0)}{1} + 2c\left(\frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{n} + \frac{1}{n+1}\right)$

$\simeq 2c(H_{n+1} - 1) \simeq c' \log n$

(where, $H_n = 1 + \frac{1}{2} + \dots \frac{1}{n} = \ln n + c$)

$\therefore T(n) = c'(n+1)\log n$

$\therefore \boxed{T(n) = O(n \log n)}$

---

6 B) i) Let take $n = 2^m$

$$T(2^m) = 2T(2^{m-1}) + 2^m \log_2(2^m)$$

$$= 2T(2^{m-1}) + m2^m$$

Calling $T(2^m)$ as f:

$$f(m) = 2f(m-1) + m2^m$$

$$= 2(2f(m-2) + (m-1)2^{m-1}) + m2^m$$

$$= 4f(m-2) + (m-1)2^m + m2^m$$

$$= 4(2f(m-3) + (m-2)2^{m-2}) + (m-1)2^m + m2^m$$

$$= 8f(m-3) + (m-2)2^m + (m-1)2^m + m2^m$$

Proceeding on the line

$$f(m) = 2^m f(0) + 2^m(1+2+\dots+m)$$

$$= 2^m f(0) + \frac{(m+1)m}{2} 2^m$$

$$= 2^m f(0) + m(m+1)2^{m-1}$$

$$T(n) = nT(1) + n(\log_2(n) + \log_2 n)$$

$$\Rightarrow T(n) = O(n \log^2 n)$$

ii) the master Th. provides a solⁿ of recurrence relation of form

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

for $a \geq 1$ & $b > 1$ with $f(n)$ asymptotically positive. Such recurrance occurs frequently.

Case 1) if $f(n) = O(n^{\log_b a - \varepsilon_c})$

$\varepsilon_c > 0$

$$T(n) = O(n^{\log_b a})$$

Case II) if $f(n) = \Theta(n^{\log_b a})$

$$T(n) = \Theta(n^{\log_b a} \log n)$$

Case III) if $f(n) = \Omega(n^{\log_b a + \varepsilon_c})$ $\varepsilon_c > 0$

$$T(n) = \Theta(f(n))$$

iii) $T(n) = 2T(n/2) + \frac{n}{\log n}$

$a = 2, b = 2$ $\therefore a = b, k = 1,$

$\varepsilon_c = -1$

$f(n) = n/\log n$

$T(n) = \Theta(n^4 \log n)$

$= \Theta(n \log \log n)$ //

---

7. A) ii) $f(n) = n^3$, if $0 \le n < 1000$

$= n^2$, otherwise

$g(n) = n$ if $0 \le n < 100$

$= n^2 + 5n$ otherwise

	0-100	100-10000	>10000
$f(n)$	n^3	n^3	n^2
$g(n)$	n	n^2+5n $= O(n^2)$	n^2+5n $= O(n^2)$

∴ Both $f(n)$ & $g(n)$ are taking same time for $n \le 10000$

∴ Answer is option 3 & 4 //

iii)

Big-O	little-O
i) Big-O is upper bound	i) little-O is bound is a stronger condition than big-O bound.
ii) Big-O is inclusive upper bound	ii) little-O is a strict upper bound

5 A) Using Recursion

Factorial Program

$f(n) = n\,f(n-1) \quad \forall\ n > 0$

$f(0) = 1 \quad \forall\ n = 0$

Pseudo Code

```
Factorial : (n) :-
    if n is 0
        return : 1
    return n * factorial (n-1)
```

Time complexity

- factorial (0) is only comparison (1 unit time)
- factorial (n) is 1 comparison, 1 multiplication, 1 subtraction & time for factorial (n-1)

$T(n) = T(n-1) + 3$

$T(0) = 1$

$T(n) = T(n-1) + 3$

$= T(n-1) + 6$

$= T(n-4) + 3k$

$T(n) = T(0) + 3n \quad (k = n)$

$T(n) = 1 + 3n$: Time complexity $= O(n)$