Assignment No. : 1

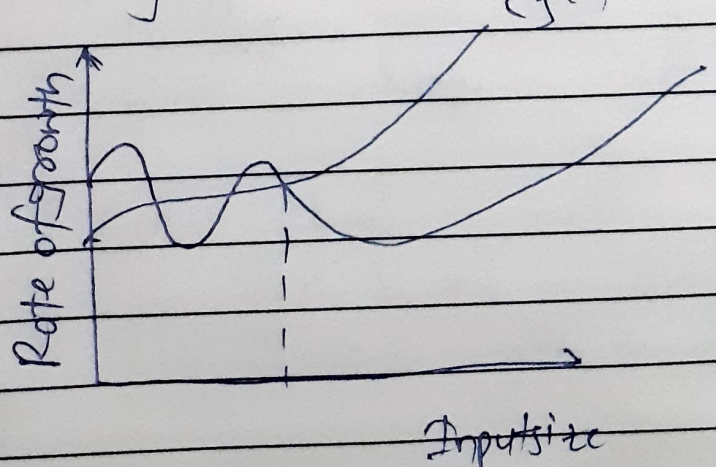
Name :- Pratiksingh Rajeshsingh Thakur
Registration No.: 2019bcs133
Roll No. : A 64
Division: A
Subject: Advanced Algorithm

Q.1) Write and explain all asymptotic notations with the help of suitable examples

Asymptotic Notation:- Mathematical notations which are used to represent complexities of algorithm are called as asymptotic notations.

Some of the asymptotic notation are:

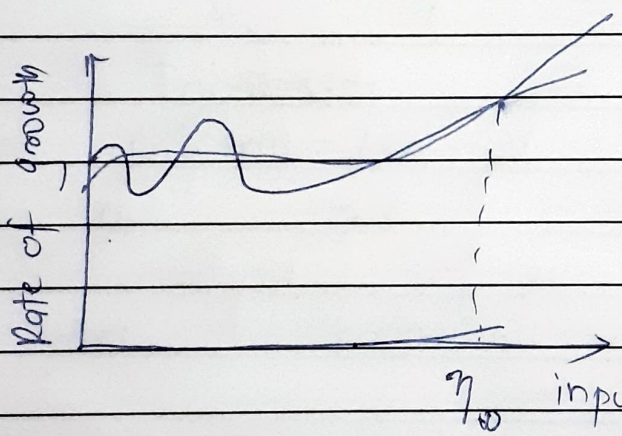
1) Big Oh Notation : Big Oh notation is used to find the upper bound of any algorithm within a constant factor.



We write $f(n) = O(g(n))$, if there are positive constant n_0 and c such that to the right of the n_0 the $f(n)$ always lies on or below $c * g(n)$

$O(g(n)) = \{ f(n) :$ There exist ⊕ve constant c such that $0 \leq f(n) \leq c(g(n)) \}$

2) Big Omega notation: Big Omega notation gives a lower bound for a function to within a constant factor.

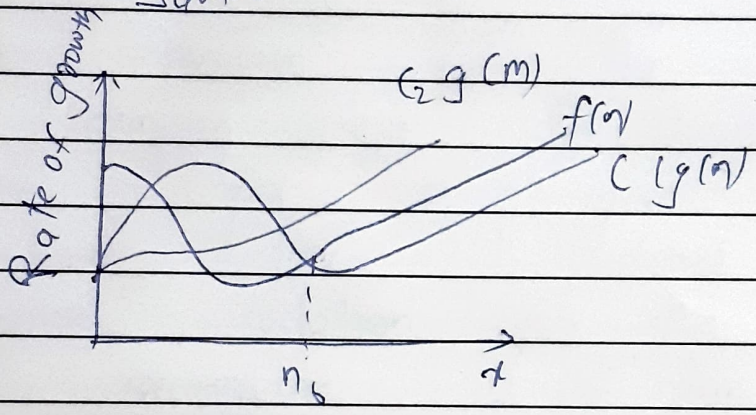


We write $f(n) = \Omega(g(n))$, if there are ⊕ve constants n_0 & c such that to the right of n_0 the $f(n)$ always lies or above $c * g(n)$

$\Omega(g(n)) = \{ f(n) :$ There exist positive constant c & n_0 such that $0 \leq cg(n) \leq f(n)$ for $n \geq n_0$

Date :

2] Big theta notation :- Big theta (Θ) notation gives bound for a function $f(n)$ to within a constant ~~force~~ factor. Thus notation decides whether the upper bound and lower bound of a given func^n are same.



We write $\Theta(g(n)) = f(n)$, if there are positive constants n_0 and c_1 & c_2 such that to the right of n_0, then $f(n)$ always lies between $c_1 g(n)$ & $c_2 g(n)$ exclusive

$\Theta(g(n)) = \{ f(n)$: There exist positive constant c_1, c_2, n_0 such that $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$, for all $n \geq n_0 \}$

for e.g :- If $f(n) = \frac{n^2}{2} - \frac{n}{2}$ then for all $n \geq 2$

$$\frac{n^2}{8} \leq \frac{n^2}{2} - \frac{n}{2} \leq n^2$$

Q2 Write and explain Master theorem for solving recurrences. Also explain each case of Master theorem with the help of suitable examples

⇒ Master theorem can be used to determine the running time of divide and conquer algorithms for a given program first we try to find the reccurance relation for the problem. If the recurrence is of the below form then we can directly given the answer without fully solving it.

Master Theorem:

statement:- If the recurrance is of the form $T(n) = aT(n) + \Theta(n^k \log^p n)$ where $a \geq 1$, $b > 1$ $k \geq 0$ & p is a real number then:

case I: If $a > b^k$ then $T(n) = \Theta(n^{\log_b a})$

e.g. $T(n) = 5T\left(\frac{n}{3}\right) + n$

here $a = 5$ $b = 3$ $k = 1$ $a > b^k$

$T(n) = \Theta(n^{\log_3 5})$

$T(n) = \Theta(n^{1.64})$

case 2: If $a = b^k$

a) If $p > -1$ then $T(n) = \Theta(n^{\log_b a} \log^{p+1} n)$

b) If $p = -1$, then $T(n) = \Theta(n^{\log_b a} \log \log n)$

c) If $p < -1$, then $T(n) = \Theta(n^{\log_b a})$

e.g $T(n) = 3T(n/3) + n^2$

here $a = 3$ $b = 3$ $k = 2$, $p = 0$

also $a = b^k$ and $p > 0$

$T(n) = \Theta(n^2 \log n)$

case 3: If $a < b^k$

a) If $p \geq 0$ then $T(n) = \Theta(n^k \log^p n)$

b) If $p < 0$, then $T(n) = O(n^k)$

e.g $T(n) = 3T(n/4) + n^2$

here $a = 3$ $b = 4$ $k = 2$ $p = 0$

now $a < b^k$ & $p = 0$

hence case 3(a) fits

$\therefore T(n) = \Theta(n^k \log^p n)$

$T(n) = \Theta(n^2)$

Q3 Write and explain deterministic & randomised versions of quicksort algorithm Also provide the proofs for respective time complexities. Sort following sequence in ascending order by applying deterministic approach for quick sort. Write each step of your solution for each clearly and show pictorically the changes of array elements partitions etc.

Input array : {30, 1, 35, 4, 40, 45, 50}

Ans ⇒

Deterministic quick sort algorithm:—

In a deterministic quick sort the pivots are chosen by either always choosing the pivot at a particular position using the median of any number of predetermined element choices for instances, a common method is to choose the median of first, last or middle element as the pivot.

Time complexity: O(n)

• Randomise Quick sort: It is a type of quick sort algorithm in which one can choose a random number as a pivot There is still the possibility of time complexity of $O(n^2)$ but the prob.

is much smaller

Proof of time complexity:

let us assume $T(n)$ be the time complexity of quick sort and also assume that all elements are different

If the pivot is the ith smallest element then exactly $(i-1)$ items will be at left of the pivot & rest will be at right

Best case: Each partition split array in halves & gives

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

$= \Theta(n \log n)$ ... using master's theorem

worst case: Each partition gives unbalanced splits & we get

$T(n) = T(n-1) + \Theta(n) = \Theta(n^2)$

This case occours when the list is already sorted and last element chose as pivot.

Average case: In the average case of Quick Sort, we do not know where the split

happens. For this reason we take all possible values of split location, add all their complexities and divide with n to get the average case complexity

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

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

in best case $T(n-i) = T(i-1)$

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

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

$$n\,T(n) = 2 \sum_{i=0}^{n-1} T(i) + n^2 + n \quad \text{---(1)}$$

Same formula for $n-1$

$$(n-1)\,T(n-1) = 2 \sum_{i=0}^{n-2} T(i) + (n-1)^2 + (n-1) \quad \text{---(2)}$$

substratit eq^n (2) from eq^n (1)

$$n\,T(n) = (n+1)\,T(n-1) + 2n$$

Dividing bothe sides $n(n+1)$

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

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

$$\vdots$$

$$= O(1) + 2\sum_{i=3}^{n} \frac{1}{i}$$

$$= O(1) + \Theta(2 \log n)$$

$$\frac{T(n)}{n+1} = O(\log n)$$

$$T(n) = O((n+1)\log n)$$

$$T(n) = O(n \log n)$$

Sorting the given sequence in ascending order.

Input array: {30, 1, 35, 4, 40, 45, 50}

step (1) :- let us select 4th element as pivot element

∴ by doing all comparisions all no.s > 4 should at rt of 4 & rest should be at left of 4

{1 4 35 30 40, 45, 50}

Step 2: now all elements at left are smaller than 4 & at rt. of 4 are greater than 4 let us sort them

step 3: At left side let us take 1 as the pivot element. no in the left one there is only one element which is already sorted.

step 4: At right side let us take 40 as pivot. now all elements greater than 40 should be at rt. of 40 & rest at left of 40

∴ {1, 4, 35, 30, 40, 45, 50}

now

step 5: now the elements at left of 40 are smaller than 40 then taking 35 as pivot. Therefore after comparision all elements greater than 35 should be at rt. of it & rest at left of it

∴ {1, 4, 30, 35, 40, 45, 50}

step 6: All the elements at left of 40 are now sorted. let us move

to the right part of 45. let us take 45 as pivot. Now doing comparision all the elements < 45 are shifted at left of 45

∴ array = {1, 20, 30, 35, 40, 45, 50}

Q4 Explain the prune search paradigm for deterministic kth selection problem and prove its average time complexity is O(n).

Ans⇒ Prune and search is a method solving optimisation problems suggested by Niomod Megiddo in 1983. The basic idea of the method is a recursive procedure in which at each step the input size is reduced by a const factor $0 < p < 1$

Prune and search algorithm to find kth smallest element:

Input: A set of n elements & k
Output: The kth smallest element in S
Approach:
Step 1: If |S| <= 5, apply on Brute force method
Step 2: Divide S into n/5 sublists each containing atmost 5 elements

Step 3: Sort each sublists using algorithm

Step 4: Recursively find p as median of medians of each sublists

Step 5: Partition S into S_1 & S_2 such that S_1 contains all elements less than or equal to p & S_2 contains all elements greater than p

Step 6: Now there can be 3 cases:

(a) If $|S_1| == K$ then $S_1(K)$ will be Kth smallest

(b) if $|S_1| > K$ then K^{th} smallest must be present in S_1

(c) Otherwise kth smallest must be in S_2

Time complexity:

Since each sublist consist 5 elements so sorting them will take constant time

$T(n) = T(3n) + T(n/5) + O(n)$

let $T(n) = a_0 + a_1 * n + a_2 * n^2$ where $a_1 \neq 0$

we have

$T(3n/4) = a_0 + (3/4)\, a_1 * n + (9/16)\, a_2 * n^2 + \dots$

$T(n/5) = a_0 + \left(\frac{1}{5}\right) a_1 * n + \left(\frac{1}{25}\right) a_2 * n^2 + \dots$

$T\left(\left(\frac{3n}{4} + \frac{n}{5}\right)\right) = T\left(\frac{19n}{20}\right) = a_0 + \frac{19}{20} a_1 n + \left(\frac{361}{400}\right) an^2 + \ldots$

$T(n) = T(3n/4) + T(n/5) + O(n)$

$T(n) = T\left(\frac{19n}{20}\right) + cn$

∴ We will get $T(n) = O(n)$

Q.5) Write and explain, Randomised ith selection problem & prove its average time complexity is $O(n)$

⇒ Randomised selection algorithm :—

```
Randomised selectf(A, p, r, i)
    if (p==r) then return A[p];
      q = Randomised_partition (A, p, r);
      j = q - p + 1
      if (i==j) then return A[q]
      else if (i<j) then return randomised select
        else random select (A, q+1, r, i-j)
Randomised_partition (A, p, r)
    i = random (p, r)
    Swap (a[i], a[r]);
    partition (A, p, r);
```

We expect that the position of random element in the sorted seq. will be uniformly distributed between 0 & n-1 where n is length of the dataset This means there's at least a 1/2 chance that it's between N/4 and 3N/4 That mean that at least half time we get a pivot that reduce the problem to no more than 3n/4 previous problem size. Each pivot selection & filtering of the elements by the pivot takes O(n) if there are N elements remaing. Based on the above, we expect the problem to be reduced to no more than 3/4 of it size after two of these O(N) passes.

$$T(N) = T(3N/4 + 2 + O(N))$$

$$= T(3N/4) + O(N)$$

$$= cN + c\left(\frac{3}{4}N\right) + c\left(\left(\frac{3}{4}\right)^2(N)\right)$$

$$T(n) = O(N)$$