MANSI DAHIYA

190121136

DS(Vth Sem)

Design and Analysis of Algorithms

Tutorial_1

Q1: What do you understand by Asymptotic notations? Define different Asymptotic notation with examples.

Ans: Asymptotic Notations. Asymptotic notations are the mathematical notations used to describe the running time of an algorithm when the input tends towards a particular value or a limiting value.

Big-O Notation (O-notation) Big-O notation represents the upper bound of the running time of an algorithm. Thus, it gives the worst-case complexity of an algorithm.

$f(n) = O(g(n))$ iff there are two positive constants c and n0 such that $|f(n)| \leq c * |g(n)|$ for all $n \geq n0$

Example:

$n^2 + n = O(n^3)$

Here, we have

$f(n) = n^2 + n$,

$g(n) = n^3$

Omega Notation (Ω-notation) Omega notation represents the lower bound of the running time of an algorithm. Thus, it provides the best-case complexity of an algorithm.

$f(n) = \Omega(g(n))$ if there are two positive constants c and n0 such that $|f(n)| \geq c * |g(n)|$ for all $n \geq n0$.

Example:

- $n^3 + 4n^2 = \Omega(n^2)$
- Here, we have
  $f(n) = n^3 + 4n^2$,
  $g(n) = n^2$

Theta notation encloses the function from above and below. Since it represents the upper and the lower bound of the running time of an algorithm, it is used for analysing the average-case complexity of an algorithm.

$f(n) = \Theta(g(n))$ if there are three positive constants c1, c2 and n0.

Example:

$n^2 + 5n + 7 = \Theta(n^2)$ Where $f(n) = n^2 + 5n + 7$
$g(n) = n^2$
When $n \geq 1$,
$n^2 + 5n + 7 \leq n^2 + 5n^2 + 7n^2 \leq 13n^2$
When $n \geq 0$,
$n^2 \leq n^2 + 5n + 7$
Thus, when $n \geq 1$
$1n^2 \leq n^2 + 5n + 7 \leq 13n^2$
Thus, we have shown that $n^2 + 5n + 7 = \Theta(n^2)$ (by definition of Big-Θ, with n0 = 1, c1 = 1, and c2 = 13.)

Q2. What should be time complexity of –

for (i=1 to n)

{

i=i*2;

}

Ans: T(n) = O(n)

Q3. T(n) = {3T(n-1) if n>0, otherwise 1}

Ans: T(n) = 3T(n-1)

= 3(3T(n-2))

= 3^2T(n-2)

= 3^3T(n-3)

...

...

= 3^nT(n-n)

= 3^nT(0)

= 3^n

This clearly shows that the complexity

of this function is $O(3^n)$.

Q4. T(n) = {2T(n-1)-1 if n>0, otherwise 1}

Ans: T(n) = 2T(n-1) - 1

= 2(2T(n-2)-1)-1

= 22(T(n-2)) - 2 - 1

= 22(2T(n-3)-1) - 2 - 1

= 23T(n-3) - 22 - 21 - 20

.....

.....

= 2nT(n-n) - 2n-1 - 2n-2 - 2n-3

..... 22 - 21 - 20

= 2n - 2n-1 - 2n-2 - 2n-3

..... 22 - 21 - 20

= 2n - (2n-1)

T(n) = 1

Time Complexity is O(1).

Q5. What should be time complexity of –

int i=1, s=1;

while(s<=n)

{ i++; s=s+i;

printf(“#”);

}

Ans: I will go on i = 1,2,3-----k

s = 3,6,10,15----

while loop will terminate if : 1+2+3+ ------- +k

[k(k+1)/2] > n

So, k = O(sqrt n)

T(n) = O (sqrt n)

Q6. Time complexity of –

```
void function(int n)
{
int i, count= 0;
for (i=1; i*i<=n; i++)
count++
}
```

Ans: As the statement is valid for n/10 terms

So, T(n) = O(n/10)

ie: T(n) = O(n)

Q7. Time complexity of –

```
void function(int n){
 int i, j, k, count=0;
for(i=n/2; i<=n; i++)
for(j=1; j<=n; j=j*2)
for(k=1; k<=n; k=k*2)
count++
}
```

Ans: T(n) = $O(n\log^2 n)$

Q8. Time complexity of –

```
function(int n){
if(n==1) return;
for(i=1 to n){
```

for(j=1 to n)

{

printf("*");

}

}

function(n-3);

}

Ans: T(n) = O(n)

Q9. Time complexity of –

void function(int n){

for(i=1 to n){

for(j=1; j<=n; j=j+i)

printf("*")

}

}

Ans: T(n) =O (n^2)

As n times for I loop and n times for j loop.

Q10. For the functions, n^k and c^n, what is the asymptotic relationship between these functions? Assume that k >= 1 and c > 1 are constants. Find out the value of c and n0 for which relation holds.

Ans: n^k is O(c^n)

Ie: n^k <= c^n

Taking log on both sides we get;

K logn < = n logc

In this case the relation will for values of k and c to be n.

$$1 + \left(\frac{n}{a}\right)^k \leq c - 1.$$

$$c \geq \left(\frac{n}{a}\right)^k + 2.$$

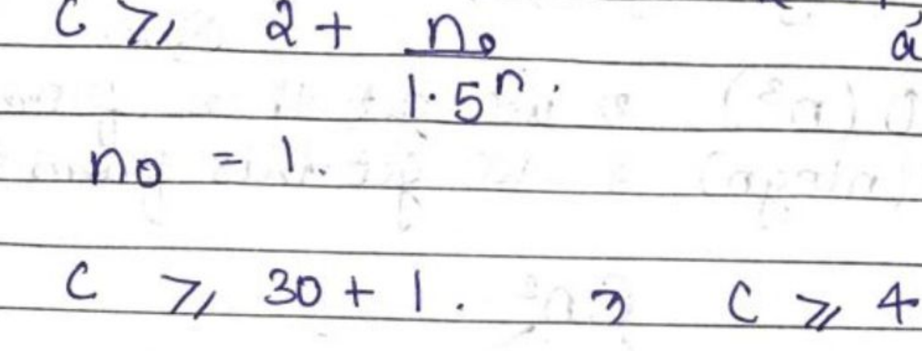
$$c \geq 2 + \frac{n_0}{1.5^n}. \qquad k = 1, \text{ let } a = 1.5.$$

$n_0 = 1.$

$c \geq 30 + 1. \Rightarrow c \geq 4.$

12. It is same as $\sqrt{n}$ $\therefore$ Ans $= O(\sqrt{n})$.

13. $T(n) = T(n-1) + T(n-2) + 1.$
Solving using tree ~~...~~ method.



T.C $\Rightarrow$ $1 + 2 + 4 + \cdots 2^n$.
This is a GP
$a = 1, \ r = 2.$

$$n = \frac{a(r^{\text{terms}} - 1)}{r-1} = \frac{2^{n+1} - 1}{2 - 1}$$

Ans T.C $= O(2^{n+1}) = O(2^n \cdot 2)$
$= O(2^n)$

Space Complexity $= O(n)$.

13 $O(\log(\log n))$ $\Rightarrow$ we get this from Q7.
$O(n^3)$ $\Rightarrow$ we get this from Q8.
$O(n \log n)$ $\Rightarrow$ we get this from Q9.

14

Cn^2.

$T(n/4)$ $\quad$ $T(n/2)$.

On further breaking this down

Cn^2

$T(n^2/16)$ $\quad$ $Cn^2/4$

$T(n/16)$ $\quad$ $T(n/8)$ $\quad$ $T(n/8)$ $\quad$ $T(n/4)$.

$T(n) = Cn^2 + 5n^2/16 + 25n^2/256 + \ldots$

This is a GP with ratio $5/16$.

So, $\dfrac{n^2}{1 - \frac{5}{16}}$ $\Rightarrow$ T.C $\Rightarrow O(n^2)$.

Date______
Page______

15 This is same as question 9 : $O(n\log n)$.

16 $i \Rightarrow 2, 2^k, (2^k)^k \ldots 2^{k\log_k(\log n)}$
The last term must be less than or equal to n, we have.
$2^{k\log_k(\log n)} = 2^{\log n} \Rightarrow n$.

18 a) $100 < \log(\log n) < \log n < \sqrt{n} < n < \log(n!) < n\log n < n^2 < 2^n < 4^n < n! < 2^{2n}$

b) $1 < \log(\log n) < \sqrt{\log n} < \log(n) < \log 2n < 2\log n < n < 2n < 4n < \log(n!) < n\log n < n^2 < n! < 2(2^n)$

c) $96 < \log_8 n < \log_2 n < 5n < \log(n!) < n\log_6(n) < n\log_2 n < 8n^2 < 7n^3 < n! < 8^{2n}$.

19
```
int ls ( int a[], int n, int d)
{
    for ( i=0; i<n-1; i++)
    {
        if (a[i] == d)
            return i;
        else if ( a[i] > d)
            return -1;
    }
}
```

Ans : Time Complexity : Best ⇒ O(1)
~~Worst = O(1)~~, Average ⇒ O(n)

Space Complexity ⇒ O(1)

20

```
void insertion_s (int a[], int n).
{
    int i, j, temp;
    for (i=1; i<n; i++)
    {
        temp = a[i];
        j = i-1;
        while ( j >= 0 && a[j] > temp)
        {
            a [j+1] = a[j];
            j = j -1;
        }
        a [j+1] = temp;
    }
}
```

This is the pseudo code for iterative insertion sort.

```
void i_s (int a[], int n)
{
    if (n <= 1)
        return;
    i_s (a, n-1);
    int l = a[n-1];
    int j = n-2;
```

```
while (j >= 0 && a[j] > l).
 {
     a[j+1] = a[j];
     j--;
 }.
 a[j+1] = l;
}.
```

An online algo is the one that can process its input piece by piece in a serial fashion i.e. in the order that the input is fed to the algorithms without having the entire input availa-ble from the beginning.

Insertion sort considers one input element per iteration and produces a partial solution without considering future elements. Thus, it is online algo

2)

Sorting Algo	Time Complexity			Space Complexity
	Best	Avg	Worst	
Bubble	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$
selection	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$
Insertion	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$
Merge	$O(n\log n)$	$O(n\log n)$	$O(n\log n)$	$O(n)$
Quick	$O(n\log n)$	$O(n\log n)$	$O(n^2)$	$O(n)$
Heap.	$O(n\log n)$	$O(n\log n)$	$O(n\log n)$.	$O(1)$.

22. Stable sort → insertion, merge, bubble sort
- online sort → insertion sort
- inplace sort → bubble, selection, insertion, heap sort.

23. Iterative pseudo code for binary search.

```
int b_s (int a[], int l, int r,
            int x)
{
  while ( l <= r)
  {
       m = l - (r+l)/2 ;
       if ( a[m] == x)
            return m;
       if ( a[m] < x)
            l = m+1;
       else
            r = m-1;
  }
  return -1;
}
```

Time Complexity = Best case $O(1)$
Avg, worst : $O(\log_2 n)$

Space Complexity = $O(1)$

Date ______
Page ______

3) Recursive binary search.

```
int b_s (int a[], int l, int r, int x)
{
    if (r >= l)
    {
        mid = (l+r)/2;
    }
  if ( a[mid] == x)
          return mid;
  else if ( a[mid] > x)
        return b_s (a, l, mid-1, x);
  else
        return b_s ( a, mid+1, r, x);
  }
  return -1;
}.
```

Time Complexity ⇒ Best $O(1)$, Avg, Worst $O(\log_2 n)$

Space Complexity ⇒ Best $O(1)$, Avg, Worst $O(\log_2 n)$.

4) $T(n) = T(n/2) + 1$. is the recurrence relation for binary recursive search