DESIGN and ANALYSIS OF ALGORITHMS

ASSIGNMENT - 1 (TCS-505)

Gaurangi Tripathi
B.Tech - CSE
Section E
5th Sem.

1) What do you understand by Asymptotic notations? Define different asymptotic notations with examples.

Asymptotic Notations are languages that allow us to analyze an algorithm's running time by identifying its behaviour as the input size for the algorithm increases
i.e. These are used to tell the complexity of an algorithm when the input is very large.

These are mathematical tools to represent the time complexity of algorithms for asymptotic analysis.

There are 5 types of asymptotic notations :-

* **Big O Notation :** $\boxed{f(n) = O(g(n))}$

  g(n) is "tight" upper bound of f(n).
  - It tells us that a certain function will never exceed a specified time for any value of input 'n'.
  - It gives the worst-case complexity of an algorithm.
  - Ex : $T(n) = O(1)$; $T(n) = O(n)$; $T(n) = O(n^2)$ etc.

* **Big Omega(Ω) :** $\boxed{f(n) = \Omega(g(n))}$

  g(n) is "tight" lower bound of f(n).
  - It represents the lower bound of the running time of an algorithm.
  - It gives the best case complexity of an algorithm.
  - Ex : If a running time is $\Omega(g(n))$, then for large enough 'n', the running time is at least $k \cdot g(n)$ for some constant 'k'.

Anytriph
(E) 25

★ Theta (Θ): $\boxed{f(n) = \Theta(g(n))}$

- It gives "tight" upper and lower bound of $f(n)$.
- It is used for analyzing average-case complexity of an algorithm.
- ex: For an algorithm having $T(n) = 3n^3 + 6n^2 + 6000$ will have $\Theta(n^3)$.

★ Small o Notation:

- It is used to describe an upper bound that cannot be tight.

$$f(n) = o(g(n)) \longrightarrow f(n) < c\,g(n) \ \forall\ n > n_0 \text{ and } \forall\ c > 0$$

★ Small Omega (ω):

- It gives lower bound that cannot be tight.

$$f(n) = \omega(g(n)) \rightarrow f(n) > c \cdot g(n) \ \forall\ n > n_0 \text{ and } c > 0$$

Ques what should be the time complexity of :-

2) for(i=1 to n) {i=i*2}

$i = 1, 2, 4, 8 \ldots n$. is a GP, where, $a = 1$; $r = 2$.

$$S_n = \frac{a(r^n - 1)}{(r-1)} \text{ and } T_k = ar^{k-1}$$

$$n = 1.(2^{k-1}) = \frac{2^k}{2} \longrightarrow n = \frac{2^k}{2} \Rightarrow 2^k = 2n$$

Taking log → $k \log_2 2 = \log_2 2 + \log n$

$$k = \log n + 1$$

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

3) $T(n) = 3T(n-1)$ —①

putting $n = n-1$ in eq. ①

$$T(n-1) = 3T(n-2) \text{ —②}$$

putting value of $T(n-1)$ from eq. ② in eq. ①

$$T(n) = 3(3T(n-1)) = 3[3T(n-2)] \text{ —③}$$

Ouyilgh
Ⓔ25

putting value of $n = n-2$ in eq. 1

$$T(n-2) = 3T(n-2) \quad \text{--- (4)}$$

putting the value of $T(n-2)$ from eq. 4 in eq 3

$$T(n) = 9[3T(n-3)] \quad \text{--- (5)}$$

$$\vdots$$

$$T(n) = 3^k T(n-k) \quad \text{--- (6)}$$

$\because T(1) = 1$

$n-k = 1$

$k = n-1$

$$T(n) = 3^{n-1} T[n-(n-1)] \quad \text{--- (7)}$$

$$T(n) = 3^{n-1} T(1)$$

$$T(n) = 3^{n-1}$$

$$T(n) = \frac{3^n}{3}$$

$$3T(n) = 3^n \longrightarrow \therefore \boxed{T(n) = O(3^n)}$$

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

$T(1) = 1 \quad ; \quad T(n) = 2T(n-1) - 1 \quad \text{--- (1)}$

put $n = n-1$ in eq. 1 ;

$$T(n-1) = 2T(n-2) - 1 \quad \text{--- (2)}$$

putting value of $T(n-1)$ from eq. 2 in eq. 1 :

$$T(n) = 2[2T(n-2)-1] - 1 \quad \text{--- (3)}$$

put $n = n-2$ in eq. 1 ;

$$T(n-2) = 2T(n-3) - 1 \quad \text{--- (4)}$$

putting value of $T(n-2)$ from eq. 4 in eq. 3

$$T(n) = 2[2(2T(n-3)-1)-1] - 1$$

$$T(n) = 4[2T(n-3)-1] - 2 - 1 \quad \text{--- (5)}$$

$$T(n) = 8T(n-3) - 4 - 2 - 1 \quad \text{--- (6)}$$

$$\therefore T(n) = 2^k T(n-k) - 2^{k-1} - 2^{k-2} \dots 2^2 - 2^1 - 2^0 \quad \text{--- (7)}$$

$\because n-k = 1$

$k = n-1$

Rupinder
2025

putting value of 'k' in eq. ⑦ ;

$T(n) = 2^{n-1}\, T(1) - [2^0 + 2^1 + 2^2 \cdots + 2^{n-3} + 2^{n-2}]$ (4)

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

$T(n) = 2^{n-1} - (2^{n-1} - 1)$

$\therefore \boxed{T(n) = 1}$

5) int i = 1, s = 1; while (s <= n) { i++; s = s+i;
} printf("#");

$i = 1, 2, 3, 4 \ldots k$.

$\frac{k(k+1)}{2} \Rightarrow k^{th}$ term formula for summation.

$\frac{k(k+1)}{2} > n$

$k = O(\sqrt{n}) \longrightarrow \boxed{T(n) = O(\sqrt{n})}$

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

$i = 1, 2, 4, 8, 16 \ldots n$. is a GP where $a = 1, r = 2$.

$t_k = a(r^n - 1) = 1(2^k - 1)$

$n = 2^{k-1} \Rightarrow 2^k = 2n \Rightarrow k = \log_2(2) + \log_2(n)$

$k = \log n + 1$

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

Ruyitolpi
E 25

7) void function(int n)

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

Complexity of 'i' loop = $O(n)$
Complexity of 'j' loop = $O(\log n)$
Complexity of 'k' loop = $O(\log n)$

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

$$\boxed{T(n) = O(n \log^2 n)}$$

8) function (int n)

```
    if (n==1)
        return;
    for(i=1 to n){
        for(j=1 to n){
            printf("*");
        }
    }
    function(n-3);
}
```

complexity of 'i' loop = $O(n)$
complexity of 'j' loop = $O(n)$

$T(n) = O(n) * O(n)$

$$\boxed{T(n) = O(n^2)}$$

9) void function(int n)

```
{
    for (i=1 to n)
    {
        for (j=1 ; j<=n ; j=j+i)
            printf(" * ");
    }
}
```

Anjali
25

10) For the functions n^k and a^n, what is the asymptotic relationship b/w these functions? Assume $k >= 1$ and $a > 1$ are constants. Find out the value of c and n_0 for which relation holds.

$f(n) = n^k$; $g(n) = a^n$

$g(n)$ is tight upper bound of $f(n)$.

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

$n^k = O(a^n)$

iff. $f(n) \leq c \cdot g(n)$

$n^k \leq c \cdot a^n \ \forall \ n > n_0$ and $c > 0$

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

$\boxed{n^k = O(a^n)}$

11) Find time complexity :-

```
void fun(int n)
{
    int j=1, i=0;
    while(i<n)
    {
        i=i+j;
        j++;
    }
}
```

$i = 1, 2, 3, 4 \ldots n$.

$T_k = \frac{k(k+1)}{2}$

$n > \frac{k(k+1)}{2} \Rightarrow 2n > k^2 + k$

12) Write recurrence relation for the recursive function that prints Fibonacci series. Solve the recurrence relation to get time complexity of the program. What will be space complexity of this program.

```
int fib(int n)
{
    if(n==1)
        return;
    return fib(n-1) + fib(n-2);
}
```

$\underbrace{fib(n-1)}_{T(n-1)} + \underbrace{fib(n-2)}_{T(n-2)}$

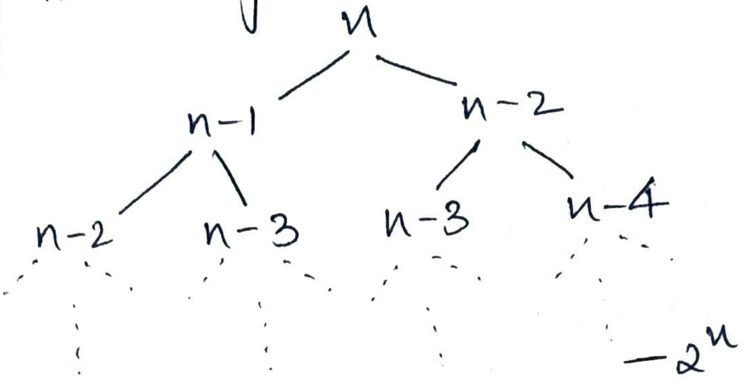
Anjali
E 25

(7)

Recursive Relation :-

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

Solving this using tree method :-



$T(n) = 1 + 2 + 4 + \cdots + 2^n$ is a GP.

$a = 1 ; r = 2$

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

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

Since, the maximum depth is proportional to 'n', hence, space complexity is $O(n)$.

13) Write programs which have the following complexities:

a) $n(\log n)$ →

```
for(int i=1; i<=n; i*=2)
{
    for(int j=1; j<=n; j+=2)
    {
        sum += j;
    }
}
```

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

$T(n)_j = O(n)$

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

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

b) n^3 →

```
for(int i=1; i<=n; i++)
{
    for(int j=1; j<=n; j++)
    {
        for(int k=1; k<=n; k++)
        {
            sum += k;
        }
    }
}
```

$\boxed{T(n) = O(n^3)}$

c) log(log n) → for(int i=2; i<=n; i=pow(i,i))

```
{
    for(int j=n; j>i; j=fun(j))
    {
        ...
    }
}
```

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

14) Solve : $T(n) = T(n/4) + T(n/2) + cn^2$

using Master's method :-

let $T(n/2) \geq T(n/4)$

$\boxed{T(n) = aT(n/b) + f(n)}$

then, $T(n) = 2T(n/2) + cn^2$

$a=2 ; b=2$

$c = \log_b a = \log_2 2 = 1$

$n^c = n^1 = n$

$f(n) = n^2 \longrightarrow f(n) > n^c \longrightarrow n^2 > n$

$\therefore T(n) = \Theta(f(n)) \longrightarrow \boxed{T(n) = \Theta(n^2)}$

15) Find complexity :-

```
int fun(int n)
{
    for(int i=1; i<=n; i++)
    {
        for(int j=i; j<=n; j+=i)
        {
            ::
        }
    }
}
```

complexity of 'i' loop = O(n)

complexity of 'j' loop = O(log n)

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

Anjali Gupta
25

16) What should be the time complexity of :-

```
for(int i=2; i<=n; i = pow(i,k))
{
    //O(1) expressions
}
```

where 'k' is constant.

Here, 'i' takes the values $2, 2^k, (2^k)^k = 2^{k^2}, 2^{k^3} \ldots$

$2^{k(\log_k(\log n))} = 2^{\log n} = n$

Therefore, we have $\log_k(\log n)$ iterations and each iteration takes a constant time to execute.

Therefore, the total time complexity is $O(\log(\log n))$

$$\boxed{TC = O(\log(\log n))}$$

17) Write a recurrence relation when quick sort repeatedly divides the array in 2 parts of 99% & 1%.
Derive the time complexity in this case.
Show the recursion tree & find heights of both the extreme parts. What do you understand by this analysis?

The partitioning scheme of 99% & 1% is one of the most unbalanced partition possible.

Time Complexity

Time Cn

$c(n-1)$

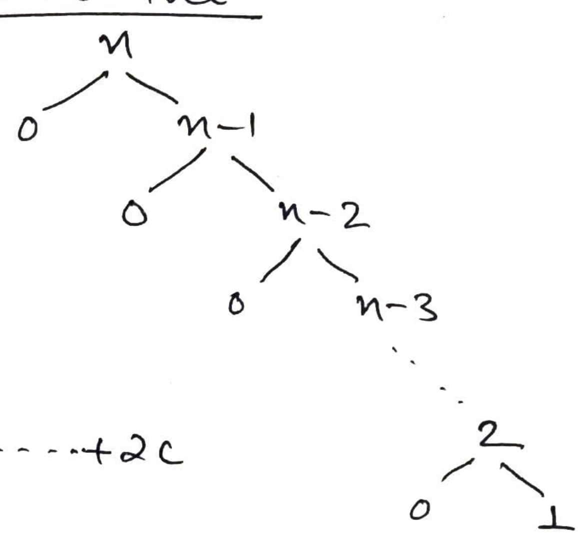
$c(n-2)$

$\vdots$

$2c$

0

Recursion Tree



Total Time: $Cn + C(n-1) + C(n-2) \ldots + 2C$

$= C(C(n+1)(n/2) - 1)$

Using big theta, ignore trivial terms

$\therefore$ Worst Case $TC = O(n^2)$

It is assumed that the original call takes (n) time where 'c' is some constant). (10)

Difference b/w entrances = n(input size).

18) Arrange the following in increasing order of rate of growth :-

a) $n, n!, \log n, \log\log n, \text{root}(n), \log(n!), n(\log n), 2^n, 2^{2n}, 4^n, n^2, 100.$

$100 < \text{root}(n) < \log\log(n) < \log(n) < n\log(n) < n < n^2 < \log(n!) < 2^n < 2^{2n} < 4^n$

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

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

19) Write pseudo code to search an element in sorted array with min. comparisons using Linear Search

```
int Linear(int *arr, int n, int key)
{
   for (i<0 to n-1)
      if (arr[i] = key)
         return i;
      return -1
}
```

20) Write pseudo code for iterative & recursive insertion sort. Insertion sort is also called online sorting. Why?

Iterative Insertion Sort :-

```
void Insertion(int arr[], int n)
{
   for i=1 to n
   int value ← arr[i];
   int j←i;
   while (j>0 & arr[j-1] < value)
   {
      arr[j+1] ← arr[j]
      j--;
   }
   arr[j] ← value;
}
```

Recursive Insertion Sort :-

```
void Insertion( int arr[], int i, int n)
{
   int value ← arr[i];
   int j ← i ;
   while (j>0 & arr[j-1] > value)
   {
      arr[j] & arr[j-1];
      j--;
   }
   arr[j] ← value ;
   if (i<=n)
   {
      Insertion(arr, i+1, n);
   }
}
```

Insertion sort is an online sorting algorithm since it can sort a list as it receives it. In all other algorithms, we need all elements to be provided to the algorithms before applying it.

Anjali
E 25

Q21) Complexity of all the sorting algorithms discussed :—

ALGORITHM	Time Complexity			Space Complexity
	Best	Avg	Worst	Worst
Bubble Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$
Insertion Sort	$O(N)$	$O(n^2)$	$O(n^2)$	$O(1)$
Merge Sort	$O(n\log n)$	$O(n\log n)$	$O(n\log n)$	$O(n)$
Heap Sort	$O(n\log n)$	$O(n\log n)$	$O(n\log n)$	$O(1)$
Quick Sort	$O(n\log n)$	$O(n\log n)$	$O(n^2)$	$O(\log n)$
Radix Sort	$O(nk)$	$O(nk)$	$O(nk)$	$O(\log n)$

Q22) Divide all the sorting algorithms into in place/stable/online sorting.

Inplace	Stable	Online
Bubble Sort	Merge Sort	Insertion Sort
Selection Sort	Insertion Sort	
Heap Sort	Bubble Sort	
Quick Sort		
Insertion Sort		

Q23) Write recursive/iterative pseudo code for binary search. What is the time & space complexity of linear & binary search (recursive & iterative)

Recursive

```
BinarySearch(arr, l, r, key)
{
    if (l >= r) return -1
        x = key
    mid = l + (r-l)/2
    if (arr[mid] == x) return mid ;
    if (arr[mid] > x)
        return BinarySearch(arr, l, mid-1, key)
```

Anjali

(E)

return BinarySearch(arr, mid + 1, r, key)
}

$\boxed{TC = O(\log n)}$ $\boxed{SC = O(\log n)}$

Iterative

```
BinarySearch(arr, l, r, x)
{
  while (l <= r)
  {
    m = l + (r - l) / 2
    if (arr[m] == x)
        return m
    if (arr[m] < n)
        l = m + 1
    else
        r = m - 1
  }
  return -1
}
```

$\boxed{\begin{array}{l} TC = O(\log n) \\ SC = O(1) \end{array}}$

24) Write recurrence relation for binary recursive search.

$$T(n) = 1 + T(n/2) + 1$$
$$= T(n/2) + c$$

so, $T(n) = T(n/2) + c$

$\boxed{TC = O(\log n)}$

Anjali
E 25