Assignment - 1

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

→ Asymptotic notations :→ They are mathematical tools to represent the time complexity of algorithms for asymptotic notations.

The main idea of asymptotic analysis is to have a measure of the efficiency of algorithms that don't depend on machine-specific constants & doesn't require algorithms to be implemented & time taken by the programs to be compared.

The following asymptotic notations are mostly used:

(1) Theta notation (Θ): The Theta notation bounds a function from above & below, so it define exact asymptotic behaviour.

(2) Big - Oh notation (O): It defines an upper bound of an algorithm; it bound a function only from above.

(3) Big Omega notation (Ω): It defines an asymptotic lower bound of an algorithm.

- Example:- Consider Insertion sort. It takes linear time in best case & quadratic time in worst case.

Thus, we can say that Insertion sort has:

- ✓ $O(n^2)$
- ✓ $\Omega(n^2)$ for worst case
- ✓ $\Theta(n)$ for best case
- ✓ $\Omega(n)$

Q2) What should be time complexity of:

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

→ i = 1, 2, 4, 8, 16, --, n

This forms a G.P.

Here, $a=1, r=2, T_k=n$

let there be 'k' terms.

$\therefore T_k = ar^{k-1}$

$\Rightarrow n = 1 \cdot 2^{k-1}$

$\Rightarrow 2^n = 2^k$

$\Rightarrow k = \log_2(2n) = \log_2(2) + \log_2(n)$

$\Rightarrow k = \log_2 n + 1$

$\therefore$ Time complexity = $O(\log_2 n + 1)$

$\Rightarrow \boxed{O(\log n)}$

Q3) Solve the recurrence relation:

$$T(n) = \begin{cases} 3T(n-1) & \text{if } n>0 \\ 1 & \text{otherwise} \end{cases}$$

→ $T(n) = 3T(n-1)$ — ①

But $n = n-1$ in ①,

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

$= 3T(n-2)$ — ②

Put ② in ①,

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

$= 9T(n-2)$ — ③

Put $n = n-2$ in (1),

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

Put (4) in (3),

$$T(n) = 9[3T(n-3)]$$
$$= 27T(n-3)$$
$$= 3^k T(n-k) \quad -(5)$$

Now $(T(1) = 1)$

$\therefore n-k = 1$

$\Rightarrow k = n-1 \quad -(6)$

Put (6) in (5),

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

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

Q4) Solve the recurrence relation:

$$T(n) = \begin{cases} 2T(n-1) - 1 & \text{if } n > 0 \\ 1 & \text{otherwise} \end{cases}$$

$\rightarrow$ $T(n) = 2T(n-1) - 1 \quad -(1)$

Put $n = n-1$ in (1),

$$T(n-1) = 2T(n-1-1) - 1 = 2T(n-2) - 1 \quad -(2)$$

Put (2) in (1),

$$T(n) = 2[2T(n-2) - 1] - 1 = 4T(n-2) - 2 - 1 \quad -(3)$$

Put $n = n-2$ in (1),

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

Put (4) in (3),

$$T(n) = 4[2T(n-3) - 1] - 2 - 1$$

$= 8T(n-3) - 4 - 2 - 1$ — (5)

$= 2^k T(n-k) - (2^k - 1)$ — (6)

Now, $T(1) = 1$

$\therefore n - k = 1$

$\Rightarrow k = n - 1$ — (7)

Put (7) in (6),

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

$= 2^{n-1} - 2^{n-1} + 1 = 1$

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

Q5) What should be the time complexity of:

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

→ $s_i = s_{i-1} + i$

If 'k' is total no. of iterations taken by program, then while loop terminates if:

$$1+2+3+\ldots+k = \frac{k(k+1)}{2} > n$$

$\therefore k = O(\sqrt{n})$

$\Rightarrow$ **Time complexity = $O(\sqrt{n})$**

Q6) What should be the time complexity of:

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

→ Time complexity = $O(\sqrt{n})$

Q7) What should be the 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 ++;
            }
        }
    }
}
```

→

i	j	k
$n/2$	$\log n$	$\log^2 n$
$n/2+1$	$\log n$	$\log^2 n$
⋮	⋮	⋮
n	$\log n$	$\frac{\log^2 n}{\frac{n}{2}\log^2 n}$

∴ Time Complexity = $O(n \log^2 n)$

Q8) What should be the time complexity of:

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

→ i: n times
j: n times
function: n/3 times

∴ Time Complexity = $O(n^3)$

Q9) Time Complexity of:

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

→ Timer loop will execute $\left(n+\frac{n}{2}+\frac{n}{3}+\dots+\frac{n}{n}\right)$ times

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

∴ Time complexity = $O(n \log n)$

Q10) for the functions, n^k & a^n, what is the asymphatic relationship between these function? Assume that $k >= 1$ and constants find out the value of 'c' & 'n_0' for which relation ~~holds~~ holds.

→ Given:

n^k $(k >= 1)$ $\quad$ a^n $(a > 1)$

Taking $k = a = 3$

$\Rightarrow n^3 \leq 3^n$

∴ We can say that $n^3 = O(3^n)$

and hence,

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

Q11) What is the time complexity of below case & why?

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

→ Time Complexity = $O(\sqrt{n})$

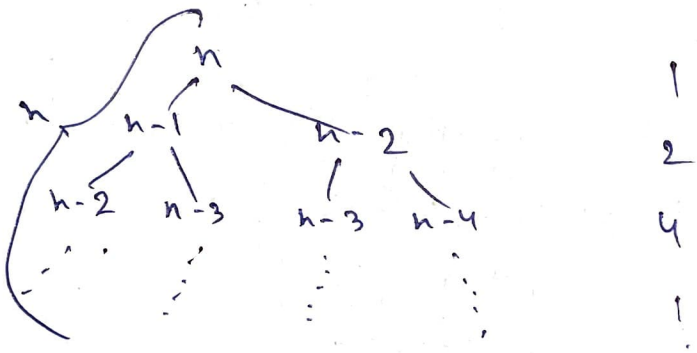
(Same logic as in Q5)

Q12) Write recurrence relation for the recursion function that prints fibonacci series.
Solve the recurrence relation to get time complexity of the program. What will be the space complexity of this program & why?

→ Recurrence relation of fibonacci Series

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

Making recurrence tree,



Time Complexity = $1+2+4+\dots+2^n$

Here, $a = 1, r = 2$

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

$$\Rightarrow O(2^{n+1}) = \boxed{O(2^n)}$$

Space Complexity = $O(n)$

This is because maximum stack frame is equal to 'n' only due to the function call:

fib(n-1) + fib(n-2)

fib(n-2) is called when the value is returned from fib(n-1)

∴ It is equal to $O(n)$.

Q13) Write Programs which have complexities:
$n\log n$, n^3, $\log(\log n)$

→ • $n \log n$

```
for (i=1; i<=n; i++)
    for (j=1; j<=n; j++)
        printf("#");
```

• n^3

```
for (i=1; i<=n; i++)
    for (j=1; j<=n; j++)
        for (k=1; k<=n; k++)
            printf("*");
```

• $\log(\log n)$

```
for (i=n; i>1; i=sqrt(i))
    printf("@");
```

Q14) Solve the following recurrence relation:
$T(n) = T(n/4) + T(n/2) + cn^2$

→ We can assume:
$T(n/2) >= T(n/4)$
$\therefore T(n) = 2T(n/2) + cn^2$

Applying Master's method, where $a=2, b=2$
$T(n) = aT(n/b) + f(n)$
$k = \log_b a = \log_2 2 = 1$

$\Rightarrow n^k = n^1 = n$
$f(n) = n^2$
$\therefore$ It is $\Theta(n^2)$

Since, $T(n) <= O(n^2)$
$\therefore$ Time Complexity $= O(n^2)$

Q16) What should be the time complexity of:

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

where 'k' is a constant.

→ $i : 2, 2^k, 2^{k^2}, 2^{k^3}, \dots, 2^{k^{\log_k \log(n)}}$

This last term must be less than or equal to 'n', & we've $2^{k^{\log_k \log(n)}} = 2^{\log n} = n$, which completely agrees with the value of our last ~~term~~ term. So, there're in total $\log_k(\log(n))$ many iteration.

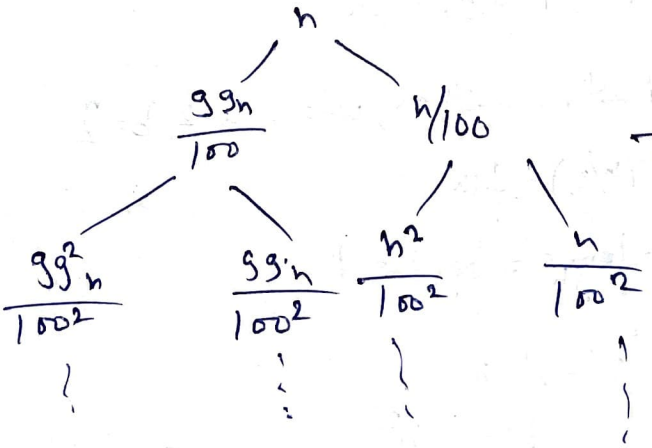
$\therefore$ **Time Complexity = $O(\log(\log(n)))$**

Q17) Write a recurrence relation when quick sort repeatedly divides the array into 2 parts of 99% & 1%. Derive the time complexity in this case. Show the recurrence tree while deriving time complexity & find the difference in heights of both the ~~extreme~~ extreme parts. What do you understand by this analysis?

→ $T(n) = T\left(\frac{99n}{100}\right) + T\left(\frac{n}{100}\right)$

Recurrence Relation

Recurrence Tree



if we take longer branch, i.e., $\frac{99n}{100}$,

Time Complexity = $\log_{\frac{100}{99}} n \approx \log n$

Thus, we can ~~easily~~ say that base of log doesn't matter as it is a constant.

Q18) Arrange the following in increasing order of rate of growth:

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

→ $100 < \log\log n, \log n < \sqrt{n} < n < \log(n!) < n\log n < n^2 < 2^n < 4^n < n!$

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

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

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

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

Q19) Write linear search pseudocode to search an element in a sorted array with minimum comparisons.

→

```
// Initially, l = 0, r = n-1
int binarysearch(int arr[], int l, int r, int x) {
    while (l <= r) {
        int m ← l + (r-l)/2;
        if (arr[m] = x)
            return m;
        if (arr[m] < x)
            l ← m+1;
        else
            r ← m-1;
    }
    return -1;
}
```

Q20) Write pseudocode for iterative & recursive insertion sort, Insertion sort is called online sorting, why? What about other sorting algorithms?

→ • Iterative Insertion sort

```
void insertsort (int arr[], int n){
    int i,temp,j;
    for (i ← 1 to n)
    {   temp ← arr[i];
        j ← i-1;
        while (j>=0 AND arr[j]>temp){
            arr[j+1] ← arr[j];
            j ← j-1;
        }
        arr[j+1] ← temp;
    }
}
```

• Recursive Insertion Sort

```
void insertionsort (int arr[], int n) {
        if (n<=1)  return;
        insertionSort (arr, n-1);
    int last ← arr[n-1];
    int j ← n-2;
while (j>=0 AND arr[j]>last){
    arr[j+1] ← arr[j];
        j ← j-1;
    }   arr[j+1] ← last;
}
```

Insertion sort considers one input element for iteration & produces a partial solution without considering future elements. So, it is called online sorting algorithm.

(Q20, 21, 22) Write complexities of sorting algorithms & also divide them into inplace / stable / online sorting.

→

Algorithm	Best Case	Average Case	Worst Case	Space Complexity	Inplace	Stable	Online
* Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$	Yes	No	No
* Bubble Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$	Yes	Yes	No
* Insertion Sort	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$	Yes	Yes	Yes

(Q23) Write recursive binary search & iterative binary search pseudocode. What is the time & space complexity of linear & binary search (Recursive & iterative).

→ • Recursive binary search

```
int binarysearch (int arr[], int l, int r, int x) {
    if (r >= l) {
        int mid ← l + (r - l)/2;
        if (arr[mid] = x)
            return mid;
        elseif (arr[mid] > x)
        return binarysearch (arr, l, mid-1, x);
        else
        return binarysearch (arr, mid+1, r, x);
    }
    return -1;
}
```

- **Iterative binary Search**

(Pseudo Code done in Q. 19)

Algorithm	Time Complexity	Space Complexity
linear Search	$O(n)$	$O(1)$
Binary Search (Recursive)	$O(\log n)$	$O(\log n)$
Binary Search (Iterative)	$O(\log n)$	$O(1)$