Design And Analysis Of Algorithms

Assignment → 01

Name → Piyush Pandey　　University Roll No → 2014996

Section → CCIS - CIS]　　Class Roll No → 23

---

1. **Asymptotic Notations** → These notations are used to tell the complexity of an algorithm when the input is large.

(i) Theta (θ) : It gives the bound in which the function will fluctuate. Theta gives the "tight upper and lower bound" both.

(ii) Big-On (O): O gives the "tight upper" bound of $f(n)$, where, $f(n) = O(g(n))$.
It implies $f(n)$ can never go beyond $g(n)$.

(iii) Omega (Ω) : $f(n) = \Omega(g(n))$
$g(n)$ is "tight lower" bound of $f(n)$.
which gives, $f(n)$ will never perform better than $g(n)$.

2. Time Complexity : for(i = 1 to n) { i = i*2 }

$\Rightarrow i = 1, 2, 4, 8, \cdots n$　　$, a = 1, r = \frac{2}{1} = 2.$

$\therefore t_k = ar^{k-1} = 1 \times (2)^{k-1}$

$\Rightarrow t_k = \frac{2^k}{2}$　$\Rightarrow 2n = 2^k.$

on taking $\log_2$;-

$k \log_2 2 = \log_2 2 + \log_2 n$

$$k = \log_2(n) + 1$$

$$k = \log_2(n)$$

$\therefore$ Time Comp. $\Rightarrow T(n) = O(\log_2(n))$ — Ans

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

$T(1) = 1$.

$\Rightarrow T(n) = 3T(n-1)$ — ①

put $n = (n-1)$ in eq ①

$\Rightarrow T(n-1) = 3T(n-2)$, put this value in eq ①.

$\rightarrow T(n) = 3[3T(n-2)] = 9.T(n-2)$

similarly, $T(n) = 27.T(n-3)$

$$T(n) = 3^k . T(n-k)$$

$\Rightarrow T(n-k) = 1$

$\therefore n = k+1$ or $k = n-1$.

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

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

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

$T(1) = 1$.

$\rightarrow T(n) = 2.T(n-1) - 1$. — ①

put $n = n-1$, then we get.

$\rightarrow T(n) = 2[2.T(n-2) - 1] - 1$

$= 2^2. T(n-2) - 2 - 1$

put $n = n-2$, then we get.

$\rightarrow T(n) = 2^3. T(n-3) - 4 - 2 - 1$

$\therefore T(n) = 2^K(T(n-K)) - 2^{n-1} - 2^{n-2} - \dots 2^{n-K} - 2^0$

$\therefore T(0) = 1$

$n - k = 1$

$n = 1 + K$ or $= k = n-1$.

$\therefore T(n) = 2^{n-1} \cdot T(1) - 2^{n-1} - 2^0$

$T(n) = 2^{n-1} - 2^{n-1} - 2^0$

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

5. Time complexity $\rightarrow$ $O(\sqrt{n})$ as $\rightarrow$ $\frac{K(K+1)}{2} = n$.

$K = \sqrt{n}$.

6. $T(n) = O(\sqrt{n})$

7. $T(n) = \frac{n}{2} \times \log(n) \times \log(n)$

$= \frac{n}{2} \cdot (\log(n))^2$

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

8. $T(n) = \frac{n}{3} \times n \times n$

$T(n) = O(n^3)$ Ans

9.

i	j
1	n
2	$n/2$
3	$n/3$
⋮	
n	n/n

$\therefore T(n) = n \times \log n$

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

10. As the growth of polynomials is slow. so, exponential has upper bound of $O(a^n)$, for $a=2$, $n_0=2$.

11. $T(n) = O(\sqrt{n})$.

12. Recurrence Relation for fibbonacci Series:-

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

Let $T(n-1) \simeq T(n-2)$

$\rightarrow T(n) = 2T(n-1) + 1$ — ①, Apply Backward substitution.

put $n = n-1$ in eq①

$\rightarrow T(n) = 2 \cdot 2 \cdot T(n-2) + 2 + 1$

put $n = n-2$ in eq①.

$\rightarrow T(n) = 2 \cdot 2 \cdot 2 \cdot T(n-3) + 4 + 2 + 1$.

$\therefore T(n) = 2^k \cdot T(n-k) + 2^k$

$\Rightarrow n-k = 1$

$k = n-1$

$T(n) = 2^n \cdot T(n-k) + 2^n$

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

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

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

```
int count = 0;
void funct1(int n)
{   for(i=1 to n, i++)
        for(j=1 to n, j=j*2)
            count++;
}
```

$T(n) = O(n^3)$

```
# void funct2(int n)
{   int count=0;
    for(i=0 to n ; i++)
        for(j=0 to n, j++)
            for(k=0 to n, k++)
                count++;
}
```

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

```
# void funct3(int n)
{   int count=0;
    for(i=n ; i>1; i=pow(i,k))
        count++;
}
```

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

Assume, $T(n/2) \geq T(n/4)$

$\therefore T(n) = 2T(n/2) + cn^2$

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

$n^k < f(n)$

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

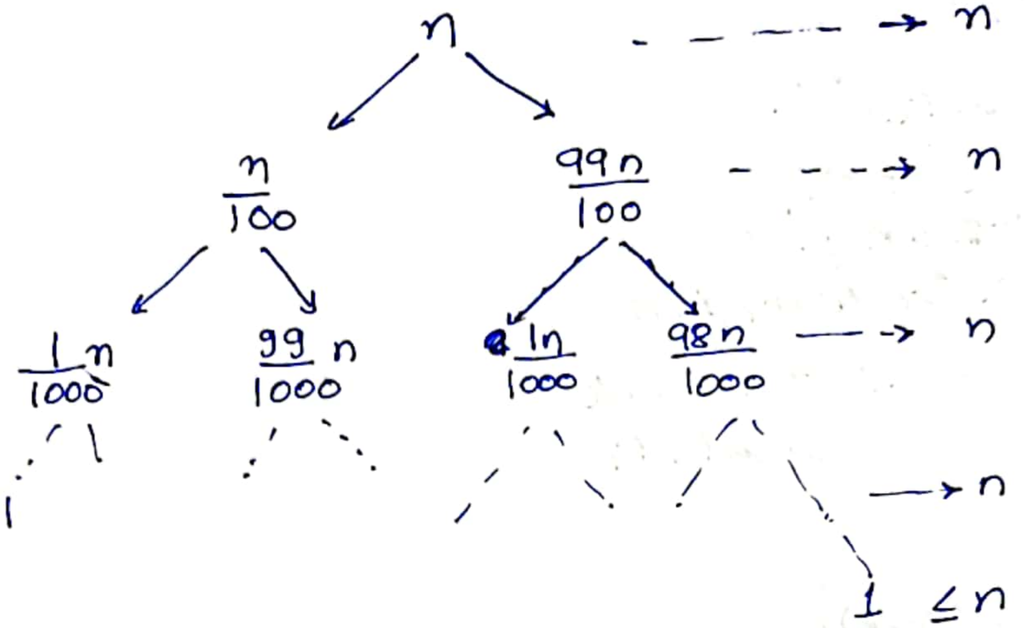
15. $T(n) = O(n \log n)$ Ans

16. $T(n) = O(\log(\log(n)))$. Ans

17. $T(n) = T(9n/10) + T(n/10) + O(n)$

for 99% and 1%

$T(n) = T(99n/100) + T(1n/100) + O(n)$



for 1%, $Tc = O(\log_{100} n . n)$ or shorter branch

for 99%, $Tc = O(\log_{100/99} n . n)$ or longer branch

either way base complexity remains same, as

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

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

(b) $1 < \log(\log n) < \sqrt{\log n} < \log n < \log 2n < 2 \log n$
$< n < n \log n < 2n < 4n = 2(2^n) < n! < n2$

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

19. Linear_Search (Arr, n, Key, flag)

```
{ Begin,
      for(i = 0 to (n-1) by 1)
      {        if (Arr[i] = key)
                   set flag = 1, break;
      }
```

```
    if flag = 1
        return flag
    else
        return -1
end }
```

20. Insertion Sort →

Iterative →

```
insertionSort( int a[], int n)
{   for(i=1 to n, by 1)
    {   value = a[i], j= i;
        while (j>0 and a[j-1]>val)
        {   a[j] = a[j-1];
                j--;
        }
        a[j] = val;
    }
}
```

Recursive →

```
insertionSort( int a[], int i, int n)
{   int val = a[i] . j = i
    while (j>0 and a[j-1] > value)
    {       a[j] = a[j-1];
                j--;
    }
    a[j] = val;
    if( i+1 < n)
        insertionSort(a, i+1, n);
}
```

- Insertion Sort is called online Sort, because on online platform, one doesnot know the input size, and we want some optimum result, which it provides.

21.

	Best	Average	Worst
Selection	$\Omega(n^2)$	$\Theta(n^2)$	$O(n^2)$
Bubble	$\Omega(n)$	$\Theta(n^2)$	$O(n^2)$
Merge	$\Omega(n\log n)$	$\Theta(n\log n)$	$O(n\log n)$
Heap	$\Omega(n\log n$	$\Theta(n\log n)$	$O(n\log n)$
Insertion	$\Omega(n)$	$\Theta(n^2)$	$O(n^2)$
Quick	$\Omega(n\log n)$	$\Theta(n\log n)$	$O(n^2)$

22.
- Bubble sort, insertion sort and selection sort are interfaced sorting algorithm.
- Bubble sort and insertion sort can be applied as stable algo. but selection sort cannot.
- Merge sort is a stable algo but not an inplace algo.
- Quick sort is not stable but is an inplace algo.
- Heap sort is an inplace algo. but not stable

23. **Iterative Binary Search :-**

```
int binarySearch (int arr[0], int l, int r, int x)
{   while( l<=r)
      {     int m = (l+r)/2;
            if (arr[m] = x)
                  return m;
            if(arr[m] <x)
                  l = m+1;

            else
                  r = m-1;
      }
      return -1;
}
```

	Tn			
It. linear.	Best → $O(1)$,	Av. $O(n)$,	Worst. $O(n)$	$O(1)$
It. Binary.	$O(1)$	$O(\log_2 n)$	$O(\log_2 n)$	$O(1)$
Recu. Binary	$O(1)$	$O(\log_2 n)$	$O(\log_2 n)$	Best → $O(1)$, Av → $O((\log_2 n)$ worst → $O(\log_2 n)$.

24. $T(n) = T(n/2) + \underline{\underline{1}}$ try