Assignment - 01

Name : Shreyansh Nanda

Section : F

University Roll no : 2014868

Class Roll no. : 52

Subject : Design and Analysis of Algorithm.

1) Asymptotic Notations : Asymptotic notations are the notations or expression that are used to represent the complexity of an algorithm.

Types:

1) Theta (Θ) : gives the bound in which the function will fluctuate. (gives Average value).

2) Big oh (O) : $f(n) = O(g(n))$

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

i.e $f(n)$ can never go beyond $g(n)$.

3) Omega (Ω) : $f(n) = \Omega g(n)$

$g(n)$ is "tight" lower bound of $f(n)$.

i.e $f(n)$ will never perform better than $g(n)$.

2) Time Complexity for:

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

$i = 1, 2, 4, 8 \ldots n$ $\quad a = 1, \; r = 2$

$$t_k = a r^{k-1}$$

$$n = \frac{2^k}{2} \Rightarrow \cancel{2^k} = 2n = 2^k$$

taking log both side

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

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

$$\Rightarrow O(\log_2(n) + 1)$$

$$\Rightarrow O(\log n)$$

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

Using Backward Substitution:

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

$$T(n-1) = 3^2 (T(n-2))$$

$$T(n-2) = 3^2 (3T(n-2-1))$$

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

$$\vdots$$

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

$$= 3^n T(0)$$

$\Rightarrow T(0) = 1$

$\Rightarrow TC = O(3^n)$

4) $T(n) = \{2T(n-1) - 1\}$ if $n > 0$, otherwise $1\}$

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

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

$T(n-2) = 2(2^2(T(n-3) - 1) - 2 - 1$

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

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

$= 2^4 (T(n-4)) - 8 - 4 - 2 - 1$

$\vdots$

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

$\because T(0) = 1$

$\Rightarrow 2^n - 2^{n-1} - 2^{n-2} \dots 2^0$

$\Rightarrow 2^n - (2^n - 1)$

$TC \Rightarrow O(1)$

5) $S = 1, 3, 6, 10 \ldots n$

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

$$k^2 = n$$

$$k = \sqrt{n}$$

$$TC = O(\sqrt{n})$$

6) Time Complexity = $O(\sqrt{n})$

7) loops $\quad i \quad j \quad k$

$\quad n/2 \quad \log n \quad \log n$

$TC \Rightarrow n/2 \times \log n \times \log n$

$\Rightarrow O(n \cdot (\log^2 n)^2)$

8) outerloop $\quad i \quad j$

$\downarrow \quad \downarrow \quad \downarrow$

$n/3 \quad n \quad n$

$TC = O(n^3)$

9)

i	j
1	n times
2	$n/2$ times
3	$n/3$ times
⋮	⋮
n	n/n times

$TC = O(n \log n)$

10) Since polynomials grow slower than exponentials n^k has an asymptotic ~~Not~~ upper bound of ~~O~~(a^n) $O(a^n)$. for $a = 2$, $n_0 = 2$

11)

i	j
1	2
3	3
6	4
10	5
	⋮

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

$$k^2 = n$$

$$k = n$$

$$TC = O(\sqrt{n})$$

12) $T(0) = 0$ $T(1) = 0$ $T(n) = T(n-1) + T(n-2) + 1$

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

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

Using backward solution

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

$$= 4(T(n-2) + 3$$

$T(n-2) = 2T(n-3)+1$

$T(n) = 2(2(2(T\,n-3)+1)+1)+1)$

$= 8T(n-3)+3$

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

$T(0) = 0$

$n - k = 0$

$n = k$

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

$= 2^n + 2^n = 1$

$TC = O(2^n)$

13) $(n \log n)$

```
Void func (int n)
{ for (i=1; i<=n ; i++)
  { for (j=1 ; j<=n ; j=j*2)
     {
        //some O(1) task
     }
  }
}
```

(n^3)

```
Void func (int n)
{
    for (i=1 to n)
    { for (j=1 to n)
        {   for (K=1 to n)
            {   //some O(1) task
            }
        }
    } }
```

$(\log(\log(n)))$

```
Void func (int n)
{
    for (i=n ; i>1 ; i= pow (i, k))
    {
        //some O(1) task
    }
}
```

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

Assume $T(n/2) >= T(n/4)$

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

$C = \log_b a$

$= \log_2 2 = 1$

$\therefore n^c < f(n)$

$TC = O(n^2)$

15)

i	j
1	n times
2	n/2 times
3	n/3 times
⋮	⋮
n/n	n/n times
	log n

$\therefore TC = O(n \log n)$

16) $i = 2, 2^k, (2^k)^k, (2^{k^2})^k = 2^{k^3} \ldots 2^{k \log k (\log(n))}$

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

$2^{\log(1)} = 1$

$\Rightarrow TC = O(\log(\log(n)))$

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

taking one branch 99% and other 1%

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

1^{st} Level $= n$

II^{nd} Level $= 99n/100 + n/100 = n$

So II remains same for any kind of partition

$\therefore$ If we take longer branch $= O(n \log_{100/99} n)$

for Shorter Branch $= \Omega(n \log_{10} n)$

either way base Complexity of $O(n(\log n))$ remains.

18) (a) $100 < \sqrt{n} < \log(\log n) < \log n < n < n \log n <$

$\log n! < n^2 < n! < 2^n < 4^n < 2^{2^n}$

(b) $1 < \log(\log n) < \sqrt{\log n} < \log n < \log 2n < n < n\log n = \log(1)$

$< 2n < 4n = 2(2^n) < n! < n^2$

(c) $96 < \log_2 n < \log n! < n \log_2 n < n \log_6 n < 5n < n! <$

$8n^2 < 7n^3 < 8^{2n}$.

19) Linear search (Array, size, key, flag)

Begin

for (i=0 to n-1) by 1 do

if (Array[i] = key)

set flag = 1

Break

if flag=1

return flag

else

return -1

end

20) Iterative

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

Recursive

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

21)

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

22) Bubble sort, insertion sort & selection sort are interface sorting algo.

Bubble & insertion sort can be applied as stable algo but selection sort cannot.

Merge sort is a stable algo but not an implace algo.

Quick sort is not stable but is an inplace algo.

Heap sort is an inplace algo but not stable.

23) int binary (int [] A, int x)

```
{
  int Low = 0 , high = A.length - 1;
  while ( Low <= high)
  {
    int mid = (Low + high) / 2;
    if ( x == A[mid] )
      return mid;
    else if ( x < A[mid] )
        high = mid - 1;
    else
        Low = mid + 1;
  }
  return -1;
}
```

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