ASSIGNMENT - 01

Name : Girija Rathore
Section : CCIS
University Roll No : 2015016
Class Roll No : 05

1. Asymptotic Notations : expressions that are used to represent the complexity of an algorithm.

Classified in the following types:

(i) Theta (Θ) : gives the bound in which the function will fluctuate. or average value.

(ii) 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 $g(n)$.

(iii) 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 $g(n)$.

Girija Rathore

2. Time Complexity:

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

$\rightarrow$ $i = 1, 2, 4, 8 \ldots\ldots n$

$a = 1 \quad r = 2$

$t_k = ar^{k-1}$

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

Taking logarithm on both sides

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

$k = \log_2(n) + 1$

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

$\Rightarrow O(\log n)$

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

Using backwards substitution:

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

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

Ginjal Rathore

$$T(n-2) = 3^2(3T(n-2-1))$$
$$= 3^3 T(n-1)$$
$$\vdots$$
$$3^n(T(n-n))$$
$$= 3^n T(0)$$

$$T(0) = 1.$$

Time Complexity $= O(3^n)$

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

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

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

$$T(n-3) = 2(2^3(T(n-4)-1)-4-2-1$$
$$= 2^4(T(n-4)) - 8 - 4 - 2 - 1$$
$$\vdots$$
$$2^n(T(n-n)) - 2^{n-1} - 2^{n-2} \cdots 2^0$$

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

Time complexity $= O(1)$

Girija Rathore

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

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

$$k^2 = n$$

$$k = \sqrt{n}$$

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

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

7.

loops	i	j	k
	$n/2$	$\log_3 n$	$\log_3 n$

Time Complexity $= n/2 \times \log_3 n \times \log_3 n$

$= O(n(\log_3^2 n)^2)$

8. Outer loop $= n/3$

'i' loop $= n$

'j' loop $= n$

Time Complexity $= O(n^3)$

Girija Rathore

9. | 'i' loop | 'j' loop |
|---|---|
| 1 | n times |
| 2 | n/2 times |
| 3 | n/3 times |
| ⋮ | |
| n | n/n times |

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

10. Polynomials grow slower than exponentials. Hence, n^k has an asymptotic upper bound of $O(a^n)$ for $a = 2, n_0 = 2$.

11. | 'i' loop | 'j' loop |
|---|---|
| 1 | 2 |
| 3 | 3 |
| 6 | 4 |
| 10 | 5 |
| | ⋮ |

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

$$k^2 = n$$

$$k = \sqrt{n}$$

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

Girija Rathore

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 substitution,

$T(n) = 2.2(T(n-2) + 1) + 1$

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

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

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

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

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

$T(0) = 0$

$n - k = 0$

$n = k$

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

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

Time Complexity $= O(2^n)$

Giriya Rathore

13. $(n \log n)$

```
void function_01 (int n)
{
    for(int i = 1; i<=n; i++)
    {
        for(int j= 1 ; j<= n ; j= j*2)
        {
            [O(1) task]
        }
    }
}
```

(n^3)

```
void function-02 (int n)
{
    for (int i= 1 to n)
    {
        for (int j= 1 to n)
        {
            for (k = 1 to n)
            {
                [some O(1) task]
            }
        }
    }
}
```

Girija Rathore

$(\log(\log n))$

```
void function-03( int n)
{
    for (int i = n ; i > 1 ; i = pow(i, k))
    {
        [some O(1) task]
    }
}
```

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

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

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

$c = \log_b a$

$= \log_2 2 = 1$

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

Time complexity $= O(n^2)$

Girija Rathore

15. 'i' loop 'j' loop

'i' loop	'j' loop
1	n times
2	n/2 times
3	n/3 times
⋮	⋮
n/n	n/n times
	$\log n$.

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

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

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

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

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

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

Taking one branch 99% and other 1%

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

1st level = n

2nd level = $\frac{99n}{100} + n/100 = n$

Girija Rathore

So 3^{rd} remains same for any kind of participation

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

for shorter branch = $\Omega(n \log_{10} n)$

Either way base complexity remains of $O(n(\log n))$

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) $96 < \log_2 n < \log n! < n\log_2 n <$
$n\log_{36} n < 5n < n! < 8n^2 < 7n^3 < 8^{(2n)}$

Girija Rathore

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

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

RECURSIVE

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

Ginju Rathor

21.

	BEST	AVERAGE	WORST
Selection	$\Omega(n^2)$	$\theta(n^2)$	$O(n^2)$
Bubble	$\Omega(n)$	$\theta(n^2)$	$O(n^2)$
Insertion	$\Omega(n)$	$\theta(n^2)$	$O(n^2)$
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 algorithm.

Bubble & Insertion sort can be applied as stable algorithm but selection sort cannot.

Merge sort is a stable algorithm but not an inplace algorithm

Quick sort is not stable but is an inplace algorithm

Heap sort is an inplace algorithm but not stable.

Girija Rathore

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

Girija Rathor