Design And Analysis of Algorithms

Assignment 1

Name: Simran

Section: B

Class Roll No: 57

University Roll No: 2014888

Ans 1: Asymptotic notation describes the algorithm efficiency and performance in a meaningful way. It describes the behaviour of time or space complexity for large instance characteristics.

So, Asymptotic notations are mathematical tools to represent the time complexity of algorithms for asymptotic analysis.

There are mainly three asymptotic notations:

1. Big-O Notation (O-notation)

The Big O notation defines an upper bound of an algorithm, it bounds a function only from above. For ex, Insertion Sort. It takes linear time in best case & quadratic time in worst case.
So we can say that the time complexity of Insertion Sort is $O(n^2)$.

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

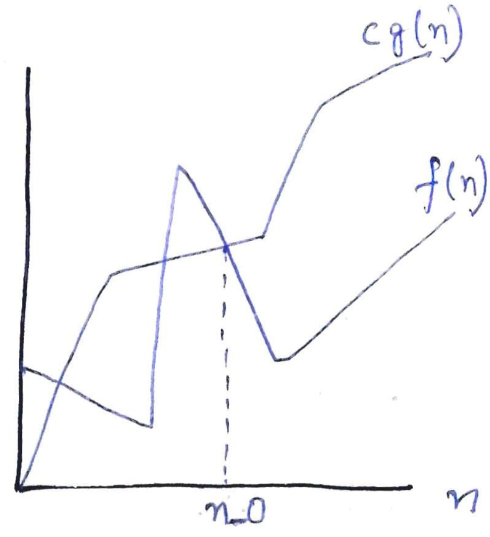
$g(n)$ is "tight" upper bound of $f(n)$

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

iff

$f(n) < c \cdot g(n)$

$\forall\ n \geq n_0$, some constant $c > 0$



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

2. **Omega Notation (Ω-notation)**

Omega notation represents the lower bound of the running time of an algorithm. It can be useful when we have lower bound on time complexity of an algorithm.

ex: The time complexity of Insertion Sort can be written as $\Omega(n)$, but is not a very useful information about insertion sort.

3. **Theta Notation (Θ notation)**

The theta notation bounds a function from above & below, so it defines exact asymptotic behavior.

Ex: $3n^3 + 6n^2 + 6000 = \Theta(n^3)$

Ans 2 ~~O(login)~~ $O(\log n)$

Ans 3 $T(n) = 3T(n-1)$ if $n>0$ otherwise 1

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

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

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

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

$\vdots$

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

Ans 4 $T(n) = 2T(n-1) - 1$ if $n > 0$ otherwise 1

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

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

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

$\vdots$

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

$\because T(0) = 1$

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

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

TC $\Rightarrow O(1)$

Ans 5. TC $= O(\sqrt{n})$

Ans 6 $O(\sqrt{n})$

Ans 7 $TC \Rightarrow n/2 \times \log n \times \log n$

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

Ans 8 $TC = O(n^3)$

Ans 9

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

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

Ans 10
Since polynomials grow slower than exponentials n^k has an asymptotic upper bound of $O(a^n)$.

for $a = 2, n_0 = 2$

Ans 11.

i	j
1	2
3	3
6	4
10	5
	⋮

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

$k^2 = n$

$k = n$

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

Ans 12

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

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

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

using backward solution

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

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

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

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

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

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

$T(0) = 0$

$n - k = 0$

$n = k$

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

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

$TC = O(2^n)$

Ans 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
    }
}
```

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

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

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

$$c = \log_b a$$

$$= \log_2 2 = 1$$

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

$TC = O(n^2)$

Ans. 15

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

$\log n$

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

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

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

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

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

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

taking one branch 99% and other 1%

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

$$1^{st} \text{ level} = n$$

$$\text{II}^{nd} \text{ level} = \frac{99n}{100} + \frac{n}{100} = n$$

So III remains same for any kind of position

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

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

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

Ans 18.

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

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

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

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

Ans 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
```

Ans. 20.

Iterative

```
insution(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

```
insution (int a[n], 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)
 insution (a, i+1, n)
}
```

Ans 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	$\Omega(n\log n)$	$\Theta(n\log n)$	$O(n\log n)$

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

Bubble & insertion sort can be applied as stable algo but ~~st~~ 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.

Ans 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 ;
}
```

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