Assignment - 1

Ques 1 Asymptotic $\xrightarrow{\text{means}}$ Towards Infinity

Asymptotic Notation are languages that allow us to analyze an algorithm's running time by identifying its behaviour as the input size of algorithm Increases. This is also known as Algorithm growth rate.

Types of Asymptotic Notations are

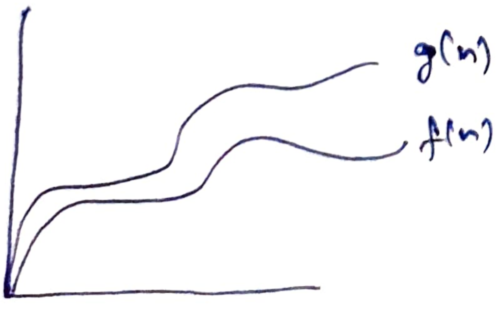
① **Big - O**

Big - O, commonly known as O, is an Asymptotic Notation for worst case, or ceiling of growth for a given function. It provides us with an asymptotic upper bound for the growth rate of the runtime of an algorithm,

Eg ① $O(n^2+n+4) = O(n^2)$ [upper limit]

② **Small - o**

Small-o commonly written as o, is an Asymptotic Notation to denote the upper bound (that is not asymptotically tight) on the growth rate of runtime of an algorithm



$f(n) = o(g(n))$ $\forall$

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

$c > 0$

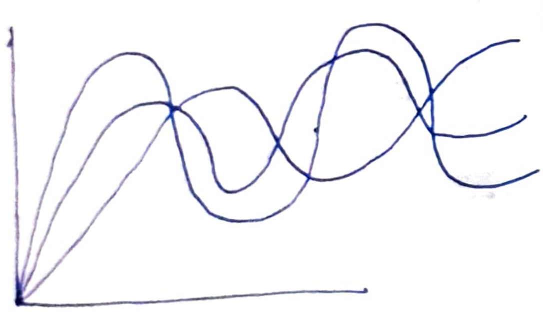
③ Big-Omega

Big-omega, commonly known as Ω, is an Asymptotic Notation for best case, or a floor growth rate for a given function. It provides us with an symptotic lower bound. for the growth rate of the runtime of an algorithm

$f(n) = \Omega(g(n))$
iff. $f(n) \geq c(g(n))$

④ Theta

Theta commonly written as Θ is an Asymptotic Notation to denote the Asymptotically tight bound on the growth rate of runtime of an algo.



$f(n) = \Theta(g(n))$
iff, $(c_1(g(n))) \leq f(n) \leq c_2(g(n))$

⑤ Small omega

Small omega, commonly written as ω, is an Asymptotic Notation to denote the lower bound (that is not asymptotically tight) on the growth rate of runtime of an algorithm

$f(n) > c \cdot g(n)$
$f(n) = \omega(g(n))$

$c > 0$

Ques 3 for(i = 1 to n)

{ i = i * 2 }

Consider, for $2^k \leq n$ (As $i = i*2$)

then, To violate this condition

$2^{k+1} \geq n$

$\Rightarrow 2^{k+1} = n \Rightarrow$ take log both side base 2

$\Rightarrow k+1 \log_2 2 = \log_2 n$

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

$\Rightarrow T(n) = O(\log_2 n)$

$\boxed{TC = O(\log_2 N)}$ $(N = n)$

Ques

$$T(n) = \begin{cases} 3T(n-1), & n > 0 \\ 1, & n \leq 0 \end{cases}$$

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

$= 3(T(0)) = 3 \times 1 = 3^1$

$T(2) = 3T(2-1) = 3T(1) = 3 \times 3 = 3^2$

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

$T(n) = 3 + 3^2 + 3^3 \ldots 3^n$

$= \frac{3(3^n - 1)}{3-1} = \frac{3}{2}(3^n - 1) = \boxed{O(3^n)}$

Ques 4

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

$T(1) = 2T(0) - 1 = 2 - 1 = 1$

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

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

Using Backward Substitution

$T(n) = 2T(n-1) - 1$ — (1)

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

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

Put value of $(n-1)$ in Eqn (1)

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

$= 2^2 T(n-2) - 2 - 1$ — (111)

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

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

Put value of $T(n-2)$ in Eqn (111)

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

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

for n in (IV) will become

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

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

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

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

$= +1$

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

Q5 Let take $n = 4$

(0) → $i = 1$ $s = 1$

(1) → $s <= n \Rightarrow 1 <= 4$

$i = 2$ $s = 2 + 1 = 3$

(2) → $s \leq n \Rightarrow 3 <= 4$

$i = 3$ $s = 3 + 3 = 6$

The s here Increases with Rate of i,

$\therefore$ $1 + 2 + 3 \ldots + k$ [k = iteration for which $s <= n$]

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

$\Rightarrow \frac{k^2 + k}{2} < n$ $\Rightarrow$ $\boxed{k^2 = O(\sqrt{n})}$

Q6 Consider for any term k the condition violates

$k * k >= n \Rightarrow k^2 = n$

$\Rightarrow \boxed{k = O(\sqrt{n})}$

Q7 The j & k loop is Independent of i loop so,

Consider, for j loop $2^k < n$

Then, $2^{k+1} \geq n \Rightarrow k+1 = \log n$

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

Consider for k loop $2^x < n$

Then, $2^{x+1} \geq n \Rightarrow x + 1 = \log n$

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

for outer most loop,

$$T(n) = \frac{n(n+1)}{4} \Rightarrow O(n)$$

Total $\boxed{TC = O(n \log^2 n)}$

Q8. TC ⇒ function → $O(n)$

loops → $O(n^2)$

$\boxed{\text{Total } TC = O(n^3)}$

Q9 Outer loop → $O(n)$

~~Inner loop → O(n)~~

~~Total TC → O(n^2)~~

The Inner loop Executes n/i times. ⇒ $O(\log n)$

Total TC → $O(n.\log n)$

Q10 Here n^k is $O(c^n)$ as for example

If we take $n = 2$ $k = 2$ $c = 2$

Then $2^2 \leq 2^2$ so, c^n is upper

limit of n^k.

Q11 ~~we can define the term i A/c to relation~~

~~seq~~

we can define the term 'i' A/c to relation

$$i_{j} = i_{j-1} + j$$

The value of j increases by 1 for each iteration. So basically the i is sum of j till j^{th} term plus initial value of i

So, $1 + 2 + 3 + 4 \dots K < n$ [As, adding i will make i Equal to n or greater]

$$\frac{K(K+1)}{2} < n$$

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

Q12 For fibonacci Series Recurrence Relation

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

$$T(2) = T(2-1) + T(2-2) + 1$$
$$= T(1) + 0 + 1 = 2^1$$

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

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

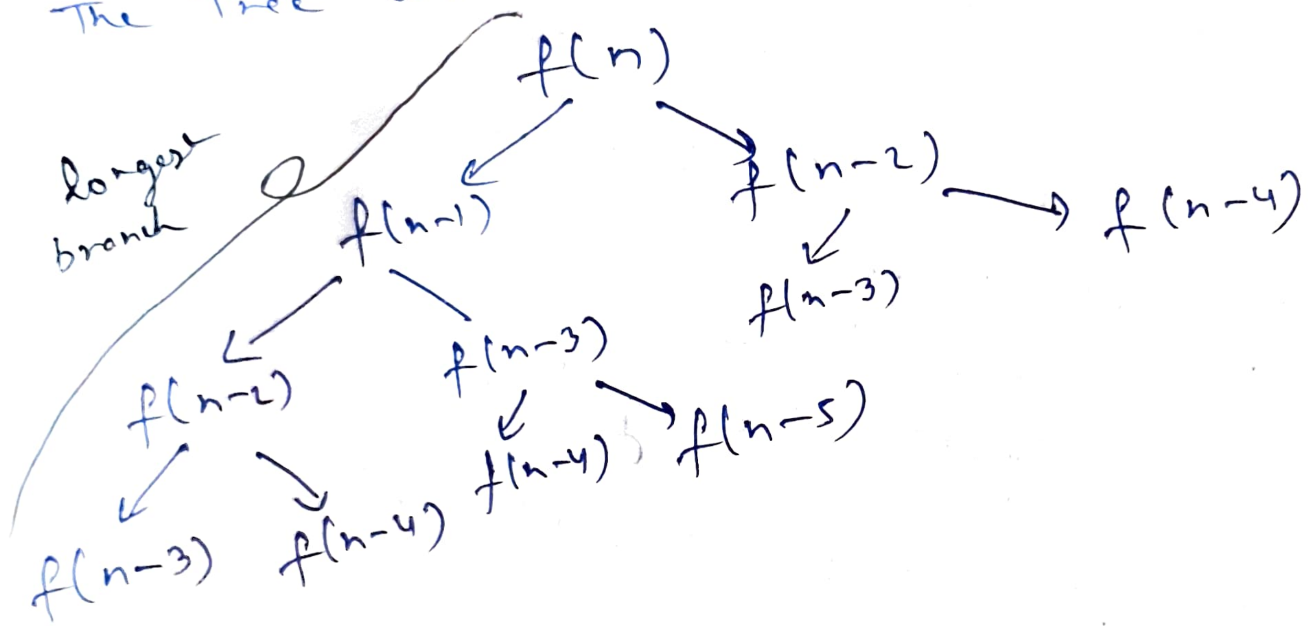
$T(= 2^0 + 2^1 + 2^2 + 2^3 + \dots + 2^n$

$= \frac{2(2^n - 1)}{1} = 2(2^n - 1)$

$\Rightarrow T(= O(2^n).$

The Space Complexity for fibonacci Series will be O(n) as there is linear call of f(n-1)

The Tree would be as follow



Q13

n log n

~~for(~~

```
for(i=0; i<n; ++i)
    for(j=0; j<n; j=j*2)
        count++
```

(n³)

```
for(i=0; i<n; ++i)
    for(j=0; j<n; ++j)
        for(k=0; k<n; ++k)
            Count++;
```

log (log n)

```
int func (int n)
{   if n==1
        return n;
    else
        return func(√n) + func(√n);
}
```

Q14 $T(n) = T(n/2) + T(n/4) + cn^2$

We can Assume that $T(n/2) \geq T(n/4)$

So we can rewrite the Eqn

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

using Master method

$a = 2 \quad b = 2$

~~[illegible]~~ $k = \log_a b = \log_2 2 = 1$

$n^k = n^1 \quad f(n) = n^2 > n^1$

So, $f(n) > n^k$

So, $TC = O(f(n)) = O(n^2)$

Q15 for outer loop we have $TC = O(n)$

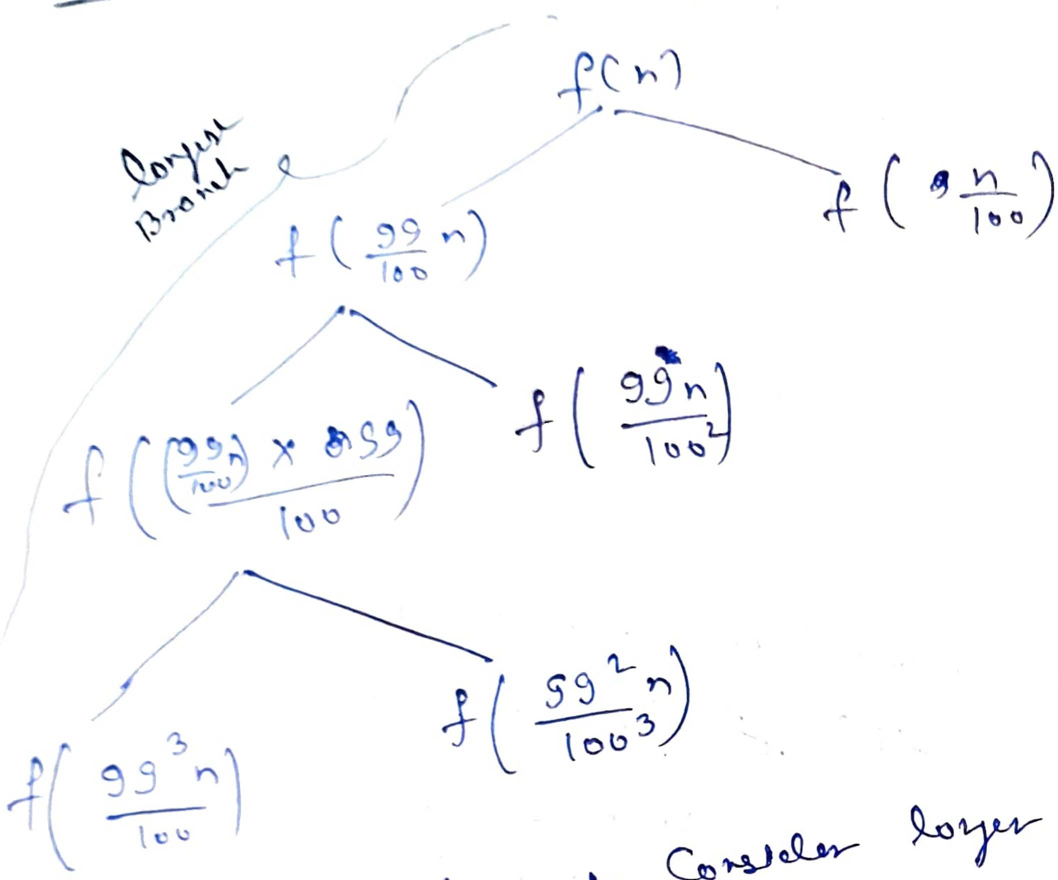
for Inner loop we have $TC = O(\sqrt{n})$

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

Q16 If K is greater than 1

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

Q17 $T(n) = T\left(\frac{99}{100}n\right) + T\left(\frac{n}{100}\right)$



we will have to consider longer branch neglecting others

$T.C = \log_{100/99} n = \log n$

Q18 (a) $100 < \log \log n < \log n < \sqrt{n} < n < \log(n!)$
$< n \log n < n^2 < 2^n < 2^{2n} < 4^n$
$< n!$

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

(c) $96 < \log_8 n < \log_2 n < 5n < \log n! < n\log_6 n$

$< n\log_2 n < 8n^2 < 7n^3 < 8^{2n} < n!$

Q19 linear Search (arr, Key)

for (i=0 to arr.size)

if arr[i] == Key

return i;

return 0;

Q20 Iterative

Insertion Sort (arr, n)

~~[illegible]~~ for(i=1 to i=n-1)

Pick element arr[i] &

Insert is into arr [0, .. i-1]

Recursive

Insertion Sort (arr, n)

{ if (n <= 1)

return

recursively Sort n-1 element

Insertion Sort (arr, n-1);

Pick last Element arr[i] and insert it

Into Sorted Sequences arr[0 ... i-1]

}

Insertion Sort considers one Input element per iteration. and produces a partial solution without considering future elements

∴ It is called Online Sorting Algorithm

Algo	Stable	Inplace	Online
Bubble Sort	✓	✓	✗
Selection Sort	✗	✓	✗
Insertion Sort	✓	✓	✓

Q21

Algo	Best	Average	Worst	S.C
Bubble	$O(n^2)$	$O(n^2)$	$O(n^2)$	1
Selection	$O(n^2)$	$O(n^2)$	$O(n^2)$	1
Insertion	$O(n)$	$O(n^2)$	$O(n^2)$	1

Q23 Iterative

linear Search (arr, n)

mid = $\frac{0+n}{2}$. low = 0 high = n

~~for~~ while (low != High)

if (arr[mid] > key)

~~low =~~ mid + 1

else if (arr[mid] < key)

~~Q23 Binary Search (arr, n)~~

~~low = 0 high = n-1~~

~~mid = (low + high)/2~~

~~arr while (low < high)~~

~~if (arr [mid] < Key)~~

Q23 Binary_Search (arr, n, Key)

```
{
    low = 0   high = n-1
    mid = (low + high) / 2
    while ( low < high ) {
        if ( arr [mid] < Key )
            low = mid + 1;
        elseif ( arr [mid] > Key )
            high = mid - 1;
        else if ( arr [mid] == Key )
            return mid
        mid = (low + high) / 2
    }
}
```

TC → Recursive → $O(\log n)$

Iterative → $O(\log n)$

SC → Recursive → $O(1)$

→ Iterative → $O(1)$

Q24 $T(n) = T(n/2) + 1$