Complexity Classes :-

(i) Constant Complexity - $O(\frac{1}{k})$ where $\frac{1}{k}$ is a constant

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

(ii) Logarithmic complexity - $O(\log_n)$

$O(\sqrt{n})$

(iii) Linear complexity - $O(n)$

$O(n \log_n)$

(iv) Quadratic complexity - $O(n^2)$

(v) Cubic complexity - $O(n^3)$

(vi) Polynomial complexity - $O(n^c)$ where c is a constant ($c > 0$)

(vii) Exponential complexity - $O(c^n)$ where c is a constant ($c > 1$)

- $(2^n < n^n)$
- $(2^n < n!)$
- $(n! < n^n)$

so, finally,

$$2^n < n! < n^n$$

$= \quad 2^n < n^n$

but, $(2^n > n^{\log n})$

i.e $n \cancel{\log 2}$ (const) $> \log n \cdot \log n$

$= \quad (n < (\log n)^{\log n})$

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

$1 < \log (\log n)$

$= \quad (n^{0.009} \log n < n)$

as $\log n \leq n^{0.991}$

$\log (\log n) \leq \cancel{0.991}$ (const) $\log n$

$= \quad (\log_2 n \not> \log_3 n)$ as $\log_3 n = \frac{\log_2 n}{\log_2 3}$

asymtotically equal but mathematical unequal

$= \frac{\log_2 n}{1.5}$

$= \quad (\log_{10} n < \log_5 n)$

$= \quad (n > \log n)$

$(n > (\log n)^2)$

i.e $\log n > 2 \log (\log n)$

$(n > (\log n)^3)$

NOTE:- If two func^n are asymtotically equal i.e differ by constant then Θ notation possible.

Q: Find True and False:

(a) $695\, n\log n = O\left(\frac{n\log n}{695}\right)$ [True]

(b) $\sqrt{\log n} = O(\log(\log n))$ [False]

(c) $0 < x < y$ then $n^x = O(n^y)$ [True]

(d) $2^n \neq O(n^c)$ where c is const $c>0$ [True]

Q: Check True or False:

(a) $(n+a)^b = O(n^b)$ where a and b are the constants T

(b) $2^{n+1} = O(2^n)$ T

(c) $2^{2n} = O(2^n)$ F

(d) $f(n) = O((f(n))^2)$ F if $f(n) = 1/n$ decreasing function

Q: Check True or False:

(a) $\frac{4^n}{2^n} = O(2^n)$

(b) $n^3 \cdot 64^{\log_2 n} = O(n^{10})$

(c) $128^{\log_2 n} \cdot 64^{\log_4 64} = \Theta(n^{10})$

(d) $f(n) = O(f(n/2))$

(a) $\frac{2^{2n}}{2^n} = 2^n = O(2^n)$ i.e True

(b) $n^3 \cdot 64^{\log_2 n} = n^3 \cdot 2^{6\log_2 n} = n^3 \cdot 2^{\log_2 n^6}$

$= n^3 \cdot n^6 = n^9$

$n^9 = O(n^{10}) =$ True

(c) $n^7 \cdot (63^7) = O(n^{10})$

$n^7 \neq \Omega(n^{10})$

so $n^7 \neq \Theta(n^{10})$ False

(d) $f(n) = 2^{2n}$ then $f(n/2) = 2^n$

so, $f(n) \neq O f(n/2)$ False

Q. Check True and False :-

(a) $100000 = O(1)$ [True]

(b) $10 = O(100000)$ [True]

(c) $10 = \Theta(100000)$ [True]

(d) $1/n = \Theta(1)$ [False]

(e) $1/n = O(1)$ [True]

Q. $f(n) = \begin{cases} n^3 & 0<n<100 \\ n^5 & n \geq 100 \end{cases}$; $g(n) = \begin{cases} n^4 & 0<n<10000 \\ n^2 & n \geq 10000 \end{cases}$

what is relation b/w $f(n)$ & $g(n)$.

after $n \geq 100$ $f(n) = n^5$ and $g(n) = n^4$

$n \geq 10000$ ↓ n^2

So, $n^5 = \Omega(n^4)$

$n^5 = \Omega(n^2)$

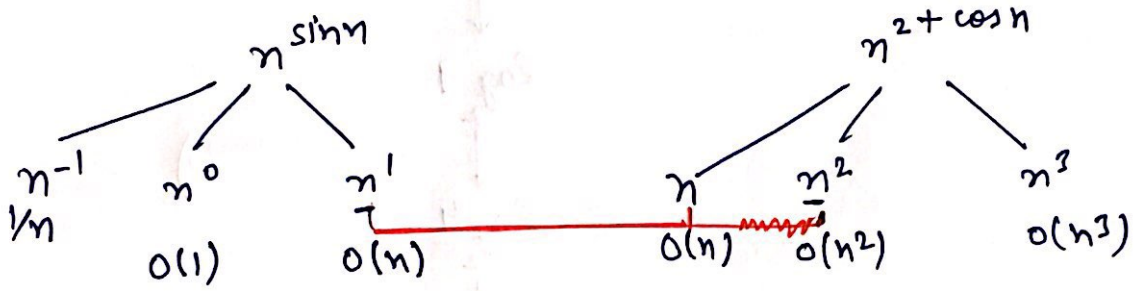
Thus $f(n) = \Omega(g(n))$

Q. $f(n) = n^{\sin n}$, $g(n) = n^{2+\cos n}$, find relation.

$f(n) = n^{\sin n}$; $g(n) = n^2 \cdot n^{\cos n}$

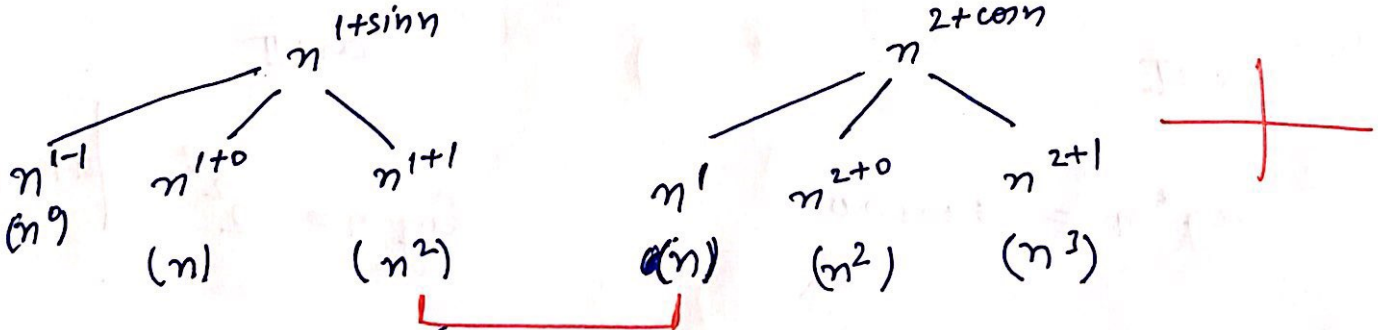
$= n^2 \cdot n^{\sin(\pi/2 - n)}$

$-1 \le \sin n \le 1$; $-1 \le \cos \le 1$



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

Q. $f(n) = n^{1+\sin n}$; $g(n) = n^{2+\cos n}$



There are chances that $f(n) \ge g(n)$ ✗

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

and since they are periodic, so we can't find any asymptotic relation because n_0 value can't determined

Q. $f(n) = \log(\log^* n)$; $g(n) = \log^*(\log n)$

Find relation b/w $f(n)$ and $g(n)$.

$\log^* n$ func^n state how many time log can be applied on n so that we remaining with 1 (termination condition)

$\log_2^* 512 = 4 \Rightarrow$

$\log_2 512$
↓
$\log_2 9$
↓ (4 times)
$\log_2 4$
↓
$\log_2 2$
↓
1

Let $n = 2^{2^{2^{\cdots^{2}}}}$] 1,00,000

Case I

$\log^* n = 100000$

$\log(\log^* n) = 17$

Case II

$\log n = 2^{2^{2^{\cdots}}}$] 99999

$\log^*(\log n) = 99999$

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

Properties of Asymptotic notations :-

(i) Reflexive - property - $f(n) = O(f(n))$

eg - O ✓, Ω ✓, Θ ✓ , o ✗, ω ✗

follow

not follow this property

(ii) Symmetric - property - if $f(n) = \Theta(g(n))$

then $g(n) = \Theta(f(n))$

eg - O ✗, Ω ✗, Θ ✓, o ✗, ω ✗

(iii) Transitive property - if $f(n) = O(g(n))$ & $g(n) = O(h(n))$, then $f(n) = O(h(n))$

eg - O ✓, Ω ✓, Θ ✓, o ✓, ω ✓

(iv) If $f(n) = O(g(n))$ then

$$h(n) \cdot f(n) = O(g(n) \cdot h(n))$$

(iv) If $f(n) = O(g(n))$ and $d(n) = O(e(n))$

then

- $f(n) + d(n) = O(\max(g(n), e(n)))$

or

$O(g(n) + e(n))$

- $f(n) * d(n) = O(g(n) * e(n))$

Q. Let $f(n)$, $T_1(n)$ & $T_2(n)$ be three +ve functions defined as follows:

$T_1(n) = O(f(n))$

$T_2(n) = O(f(n))$

then

State True / False:

(a) $T_1(n) + T_2(n) = O(f(n))$ T

(b) $T_1(n) = O(T_2(n))$ F

(c) $T_1(n) = \Omega(T_2(n))$ F

(d) $T_1(n) = \Theta(T_2(n))$ F

Q. Let $f(n)$, $g(n)$ and $h(n)$ be 3 +ve functions defined below -

$f(n) = O(g(n))$ && $g(n) \neq O(f(n))$

$g(n) = \Theta(h(n))$ && $h(n) = O(g(n))$

Then

(a) $f(n) = \Omega(h(n))$

(b) $f(n) \cdot h(n) = \Theta(g(n) \cdot h(n))$

(c) $g(n) + h(n) = \Theta(f(n))$

(d) $f(n) \cdot g(n) = O(g(n) \cdot h(n))$ ✓

Divide & Conquer :-

- Recursion - A func^n calling itself is known as recursion.

```
f(n)
{
  f(n-1);
}
```

Let us consider factorial func^n:

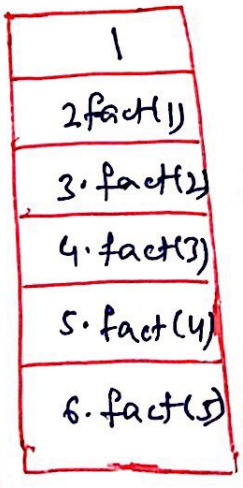
$fact(6) = 6 * fact(5)$

$5 * fact(4)$

$4 * fact(3)$

$3 * fact(2)$

$2 * fact(1)$



- Recursion is nothing but solving big problems in terms of small problems.

- To execute recursive programs we use STACK data-structure.

- Every recursive program should have a termination condition, otherwise recursion continue by pushing new call of func^{n} until stack overflow occurs.

```
f(n)
{
    if(n==1)
      return 1;
    else
      return n*f(n-1);
}
```

termination condition

recursive calls

```
f(n)
{
    s=1;
    for(int i=1; i<=n; i++)
    {
        s*=i;
    }
    return s;
}
```

Non-Recursive function

- For every recursive program, equal and non-recursive program is possible.

- Recursion program takes more STACK space compared to non-recursive program because func^n calls are more. But time-complexity will be same as logic implemented is same.

- In recursion from one func^n call to another func^n call parameter values will change, but not the no. of variables.

Recurrence Relation :-

$$f(n) = \begin{cases} 1 & \text{if } (n=1) \\ n * f(n-1) & \text{if } n>1 \end{cases}$$

$T(n) = O(n)$

Space complexity :-

(I/p + Extra)

I/p — size of input

Extra — Extra space needed by func^n calls

Non-Recursive

Space = I/p + Extra

I/p — 1-input

Extra — stack size — 1 unit

$O(1)$

Recursive

Space = I/p + Extra

I/p — 1-input

Extra — STACK size — n-units (n calls)

$O(n)$

Q: Write a recursive program and recurrence relation to find MUL of two +ve integers 'm' and 'n'.

```
mul(m,n)
{
   if(m==0 || n==0)
   {
      return 0;
   }
   else
   {
      return (m + mul(m,n-1));
   }
}
```

Recurrence Relation

$$mul(m,n) = \begin{cases} 0 & \text{if } m=0 \text{ or } n=0 \\ m + mul(m,n-1) & \text{others} \end{cases}$$

Time-complexity = $\underbrace{m + m + \cdots\cdots + m}_{n\text{-times}}$

= $O(n)$

Space-complexity = I/P + Extra

I/P → (m,n) → (2B)

Extra → STACK → n-calls → (nB)

= $O(n)$

Q. write a recursive prog & recurrence relation to find n^{th} fibonacci number.

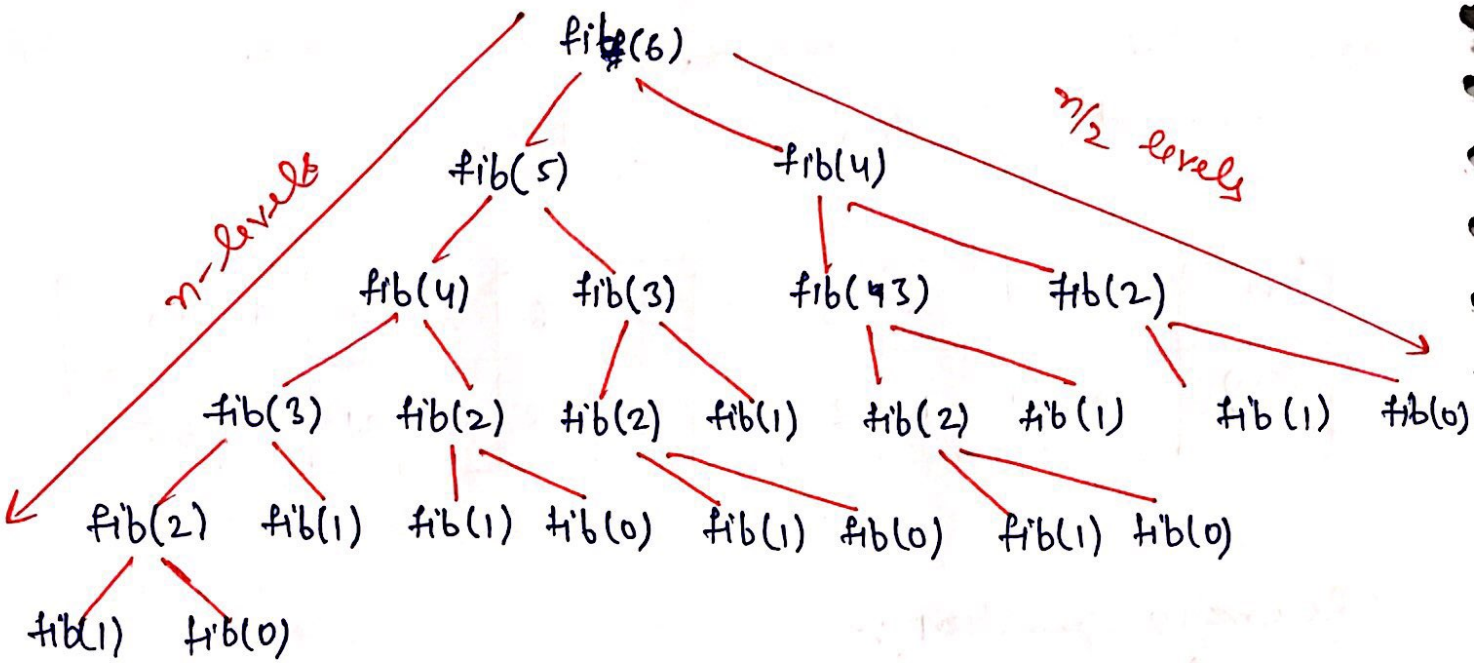
n	0	1	2	3	4	5	6	7	8	9	10
f(n)	0	1	1	2	3	5	8	13	21	34	55

Recurrence relation :-

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

Program :-

```
fib(n)
{
   if (n==0 || n==1)          // termination condition
   { return n;
   }
   else
   {
      return (f(n-1) + f(n-2));   // recursive statement
   }
}
```



$$f(n-1) \;,\; f(n-2)$$

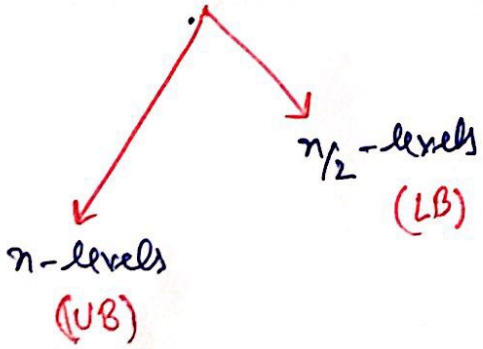
n times $\quad$ $n/2$ times

no. of nodes in a complete binary tree $= 2^n - 1$

where n is the no. of levels.

Since, due to different no. of level in LEFT & RIGHT side of a binary tree :

$$\text{no. of nodes} < 2^n - 1$$



So, time complexity is the no. of func^n calls, i.e no. of nodes present in the tree.

LB UB

So, $(2^{n/2}-1) \leq$ no. of nodes $\leq (2^n-1)$

$\Omega(2^{n/2}) \leq T(n) \leq O(2^n)$

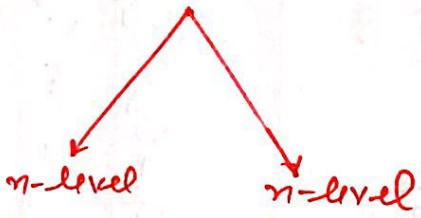
So,

$$T(n) = O(2^n)$$

&

$$T(n) = \Omega(2^{n/2})$$

if



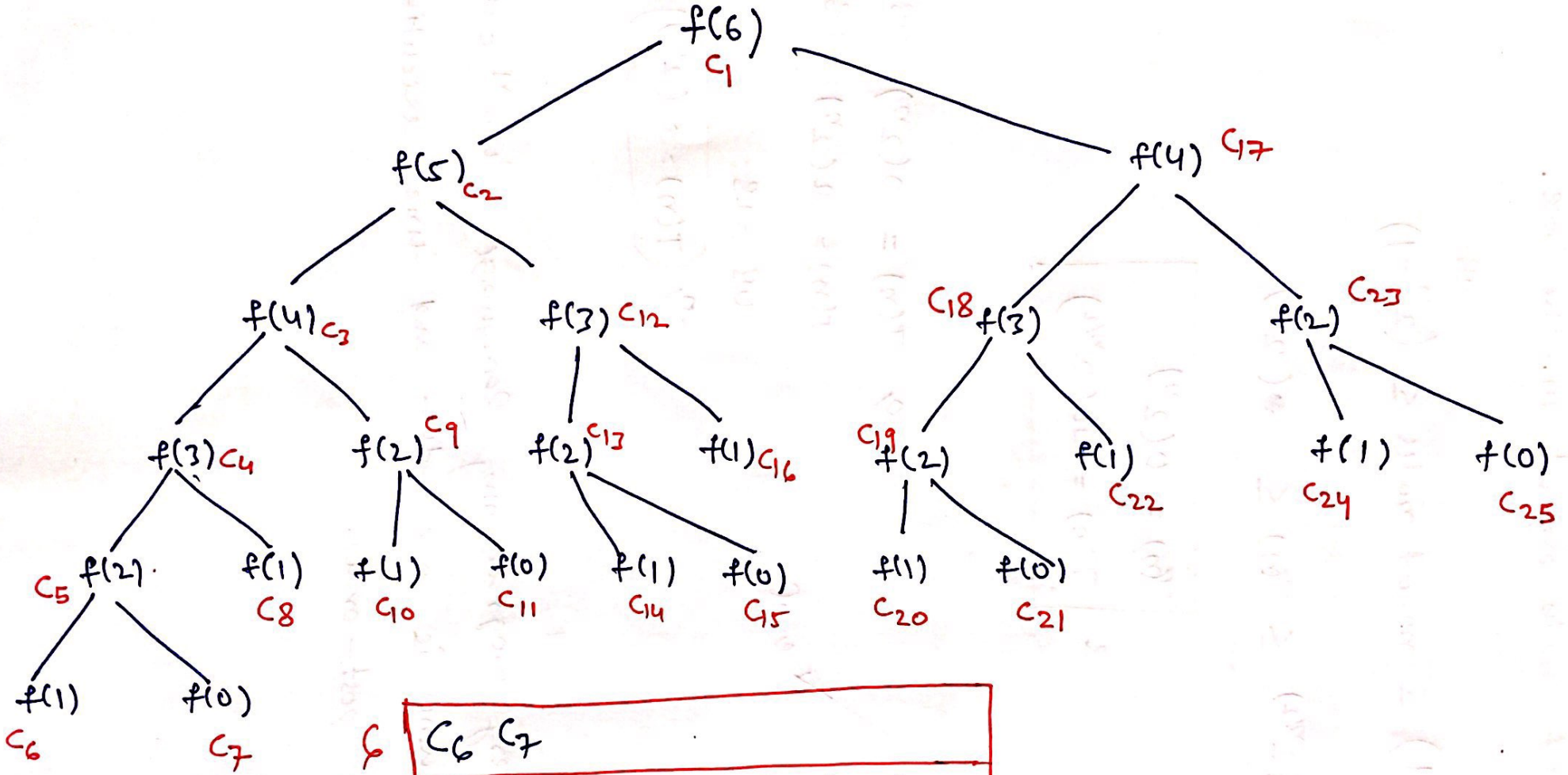
So, $T(n) = O(2^n)$ UB

$T(n) = \Omega(2^n)$ LB

UB = LB

So, $T(n) = \Theta(2^n)$

NOTE- In every programming language func^n call -ing sequence is pre-order and func^n execution sequence is post-order.



6	C_6 C_7
5	C_5 C_8 C_{10} C_{11} C_{14} C_{15} C_{20} C_{21}
4	C_4 C_9 C_{13} C_{16} C_{19} C_{22} C_{24} C_{25}
3	C_3 C_{12} C_{18} C_{23}
2	C_2 C_{17}
1	C_1

STACK

NOTE :- Recursive program STACK space based on no. of levels of recursion tree, not on no. of calls.

so, $n = 6$ and STACK - size = 6

level of tree

so, Space - complexity = $O(n)$