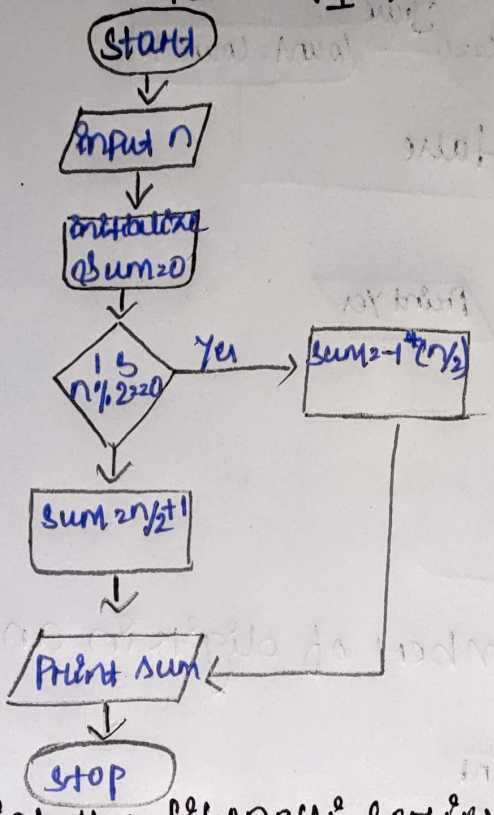
Question-01

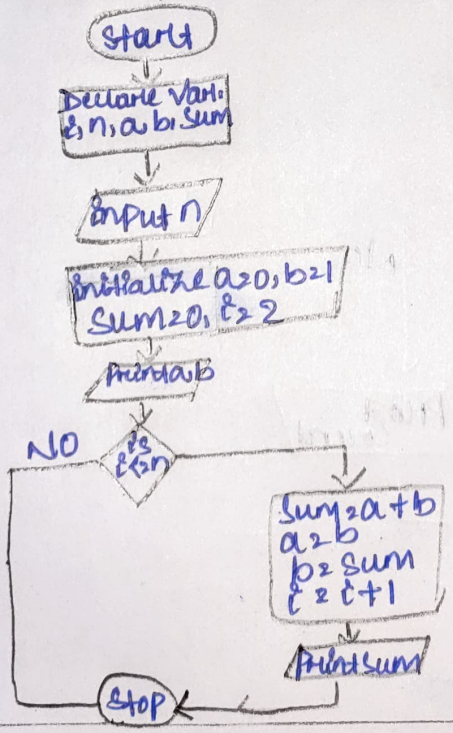
Draw the flow chart:-

a) To find the sum of the following series with n numbers starting from 1:-

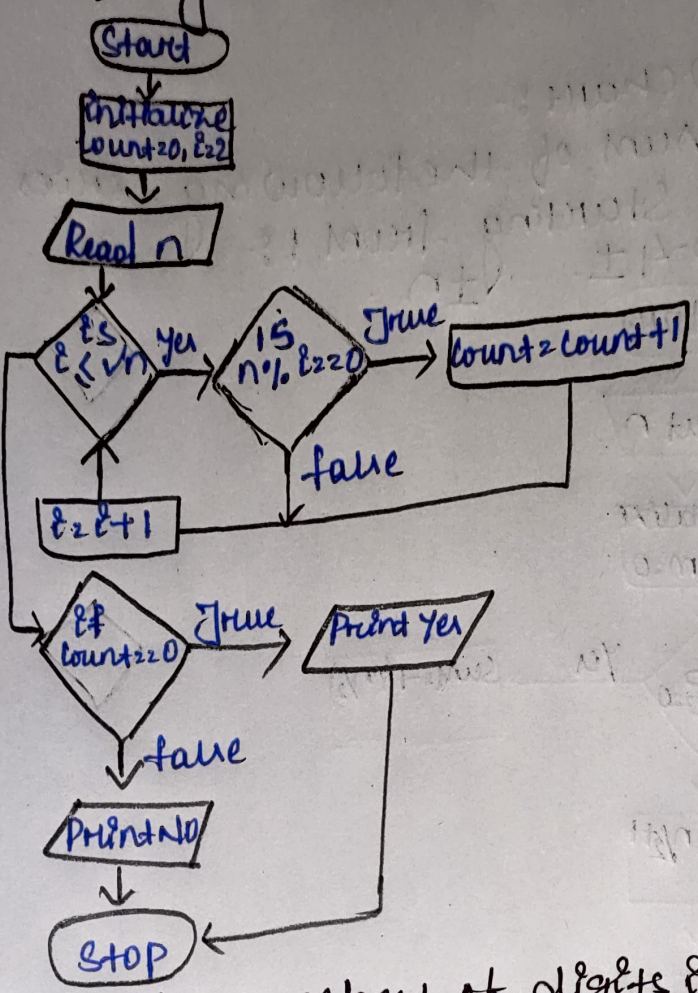
Sum = 1-2+3-4±---±n.



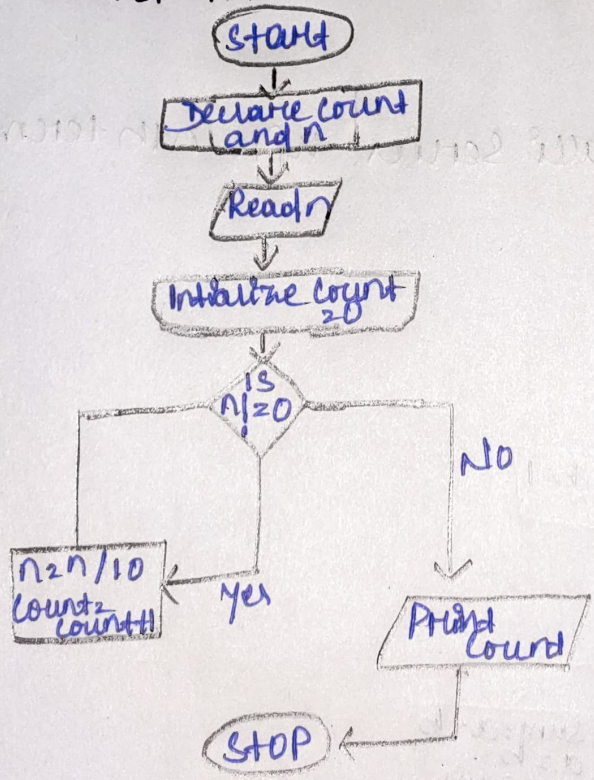
b) To print the fibonacci series upto nth term:-



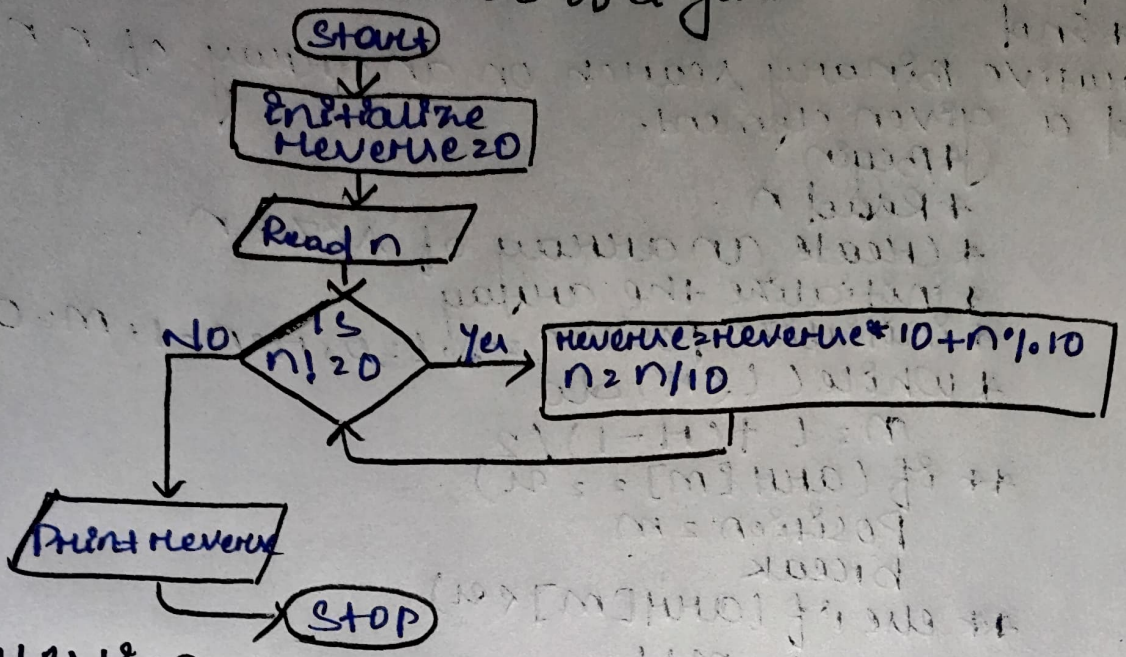
c) To check if a given number is prime or not.



d) To count the number of digits in an integer.



e) To find the reverse of a given number.



Question-02

Write Pseudocode for the following algorithm

a) To check if a number is palindrome or not.

* Begin
* Read number as n.
* Set Reverse=0, temp=n
* While n!=0 do
  Reverse = Reverse * 10 + n%10
  n=n/10
  end while.
* If n==temp
  print(n is palindrome)
  else
  Print(n is not palindrome)
* End.

b) To find the sum of the digits of a number.

* Begin
* Read number as n.
* Initialize sum=0
* While n!=0 do
  sum=sum+n%10
  n=n/10
  end while

* Print sum
* End

c) Iterative Binary search on an array of n no. and a given element.

* Begin
* Read n.
* Create an array of size n.
* Initialize the array
* Initialize position = -1, L = 0, H = n-1, M = 0
* While (L <= H) do
  M = L + (H-1)/2
** If (arr[M] == x)
  Position = M
  break
** else if (arr[M] < x)
  L = M+1
** else
  H = M-1
end while
* If (Position == -1)
  Print (Element not present)
else
  Print (Element is present at: position)

d) To convert a given decimal number to the corresponding binary number.

* Begin
* Read n
* Create an integer array binary[] of size 40
* Initialize index = 0
* While n != 0 do
  binary [index++] = n % 2
  n = n/2
* for i = index-1 to 0 do
  Print (binary [i])
  i = i-1
* End

e) To check if a number can be represented as the sum of two prime numbers or not.

```
*Begin
* Read n
* Initialize boolean possible = false.
* for i = 2 to n/2 do
      if is prime(n-i) && is prime(i)
      possible = true
      break
* if (possible = true)
      Print("NO")
* Creat a function boolean isprime(int a)
* for i = 2 to √n do
      if (a % i == 0)
        return false
    i = i + 1
  end for
  return true
* End.
```

Question-09

If the element is present in the middle, say this above array, then no comparisons are needed & element would be searched at once.

for ex:- Here l=0, h=60

mid = $\lceil \frac{l+h}{2} \rceil$ = 30, [A[mid] = = targeted element]

Suppose the element is present at the ~~by~~ beginning or at the end of the sorted array, we first compare the search element with A[n/2] & recurse either on the first half A[0, -- n/2 - 1] or 2nd half A[n/2, --, n-1].

Thus, we can find the recurrence to be T(n) = T([n/2]) + O(1). The first part of the recurrence should be obvious. The O(1) part comes because we have to determine which half of the array to recurse on. This comparison can be done in constant time.

using Master's Theorem, to solve this Problem,
$a=1, b=2, c=0$

Note:- $T(1)=1$, Let $n=2^k \Rightarrow k=\lceil \log_2 n \rceil$

$$\begin{aligned}
T(2^k) &= T(2^k/2)+1 \\
&= T(2^{k-1})+1 \\
&= [T(2^{k-2})+1]+1 \\
&= T(2^{k-2})+2 \\
&= [T(2^{k-3})+1]+2 \\
&= T(2^{k-k})+k \\
&= T(2^0)+k \\
&= T(1)+k \\
&= (1+k)
\end{aligned}$$

$T(2^k) = 1+k \Rightarrow T(n) = \lceil \log_2 n \rceil + 1$

Therefore, Performing binary search, you should expect to have $\lceil \log_2 n \rceil + 1$ comparisons where n is the no. of elements in your search space.

The previous array, there were total 60 elements, Total no. of comparisons $= \lceil \log_2 60 \rceil + 1$

$= 6+1 = (7)$

Hence, there will be 7 comparisons in the array, if it is located at beginning or end & not in the mid.

Question-03

For each of the following pairs of functions, determine whether $f(n) = O(g(n))$ or $g(n) = O(f(n))$.

a) $f(n) = n(n-1)/2$ & $g(n) = 6n$.

Let us assume:-

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

$\Rightarrow f(n) \leq C \cdot g(n)$

$\Rightarrow \frac{n(n-1)}{2} \leq C(6)(n)$

$\Rightarrow \frac{n-1}{12} \leq C$

$\Rightarrow n/12 - 1/2 \leq C \Rightarrow n/12 \leq C \Rightarrow \boxed{C \gtrsim n/12}$

When $n \to \infty$, then there doesn't exists any no. C which is greater than ∞.

So, $\Rightarrow \frac{n(n-1)}{2} \not\leq (C)(6)(n)$

$\Rightarrow f(n) \not\leq C \cdot g(n)$ (N.P)

Let us assume, $g(n) = O(f(n))$

$\Rightarrow g(n) \leq C f(n)$

$\Rightarrow [(6n)] \leq C \cdot \frac{n(n-1)}{2}$

$\Rightarrow \frac{12}{n-1} \leq C \Rightarrow \boxed{C \gtrsim \frac{12}{n-1}}$

When $n \to \infty$, then $C \gtrsim 0$, which is possible.

Hence, $C \cdot \frac{(n)(n-1)}{2} \gtrsim 6n \Rightarrow 6n \leq \frac{C \cdot (n)(n-1)}{2}$

$\Rightarrow g(n) = O(f(n))$ → Proved //

b) $f(n) = n + 2\sqrt{n}$ & $g(n) = n^2$

Let us assume:- $f(n) = O(g(n))$

$\Rightarrow f(n) \leq C \cdot g(n)$

$\Rightarrow n + 2\sqrt{n} \leq C \cdot n^2$

$\Rightarrow \boxed{1/n + 2/n^{3/2} \leq C}$ If $n \to \infty$, then $C \gtrsim 0$ which is possible.

Hence, $f(n) = O(g(n))$ → Proved //

↳(b) continuing after (c).

c) $f(n) = n + \log n$ & $g(n) = n\sqrt{n}$

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

$\Rightarrow n + \log n \leq c \cdot n\sqrt{n}$

$\Rightarrow n/n + \frac{\log n}{n} \leq \sqrt{n} \cdot c$

$\Rightarrow 1 + \frac{\log n}{n} \leq \sqrt{n} \cdot c$

As $\log n/n \to 0$, when $n \to \infty$

So, $1 \leq c\sqrt{n}$ → which is possible

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

Let us assume:- $g(n) = O(f(n))$

$\Rightarrow n\sqrt{n} \leq c \cdot (n + \log n)$

So, $g(n) \neq O(f(n))$ $\Rightarrow \sqrt{n}/c \leq 1 + \log n/n$ So, when $n \to \infty$, then $\log n/n \to 0$

$\Rightarrow \boxed{\sqrt{n}/c \leq 1}$ → which is not possible

Continuing (b)

Now, let us assume, $g(n) = O(f(n))$

$\Rightarrow g(n) \leq c \cdot f(n)$

$\Rightarrow n^2 \leq c \cdot (n + 2\sqrt{n})$

$\Rightarrow 1 \leq c \cdot (1/n + 2/n^{3/2})$

$\Rightarrow c \geq \frac{1}{(1/n + 2/n^{3/2})}$ when $n \to \infty$ then $c \to \infty$

Hence, $g(n) \neq O(f(n))$ (which is not possible)

d) $f(n) = n \log n$ & $g(n) = n\sqrt{n}/2$

Let us assume:- for $f(n) = O(g(n))$

$\Rightarrow f(n) \leq c \cdot g(n)$

$\Rightarrow n \log n \leq c \cdot n\sqrt{n}/2$

$\Rightarrow \log n \leq c\sqrt{n}$

$\Rightarrow c \geq \log n/\sqrt{n}$ when $n \to \infty$, then $\log n/\sqrt{n} \to 0$

$\Rightarrow \boxed{c \geq 0}$ → which is possible.

Hence, $f(n) \leq c \cdot g(n) \Rightarrow \boxed{f(n) = O(g(n))}$

Let us assume:- $g(n) = O(f(n))$

$\Rightarrow g(n) \leq c \cdot f(n)$

$\Rightarrow \frac{n\sqrt{n}}{2} \leq c \cdot n \log n$

$\Rightarrow c \geq \frac{\sqrt{n}}{2\log n}$, So, when $n \to \infty$, then $\frac{\sqrt{n}}{2\log n} \to \infty$

$\Rightarrow$ $c \not\geq \infty$ $\Rightarrow$ which is impossible.

Hence, $g(n) \neq O(f(n))$

e) $f(n) = 2(\log n)^2$ & $g(n) = \log n + 1$

Let us assume:- $f(n) = O(g(n))$

$\Rightarrow 2(\log n)^2 \geq O(\log n + 1)$

$\Rightarrow 2(\log n)^2 \leq c \cdot \log n + 1$

$\Rightarrow c \not\geq \frac{2(\log n)^2}{\log n + 1}$ $\Rightarrow$ when $n \to \infty$ then $c \not\geq \infty$ (N.P)

So, $f(n) \not\leq g(n) \Rightarrow f(n) \neq O(g(n))$

Let us assume:- $g(n) \leq c \cdot f(n)$

$\Rightarrow \log n + 1 \leq c \cdot 2(\log n)^2$

$\Rightarrow \frac{\log n + 1}{2(\log n)^2} \leq c$

$\Rightarrow \frac{1}{2\log n} + \frac{1}{2(\log n)^2} \leq c$

$\Rightarrow n \to \infty$, then $\log n, (\log n)^2 \to \infty$

$\Rightarrow$ $0 + 0 \leq c$ $\Rightarrow$ which is possible

So, $g(n) = O(f(n))$ $\Rightarrow$ Proved.

Question-04

State TRUE OR FALSE Justifying your answer with reason

a) $2n^2 + 1 = O(n^2)$

We can write:- $2n^2 + 1 \leq 2n^2 + n^2$

$\Rightarrow 2n^2 + 1 \leq 3n^2$

$\therefore$ Where $c = 3$ & $f(n) = n^2$

So, $\forall n \geq 1$, we have $T(n) = O(n^2)$

So, answer:- True

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

$n^2(1 + \sqrt{n}) = n^2 + n^2\sqrt{n}$

We have:- $n^2 + n^2\sqrt{n} \leq n^2\sqrt{n} + n^2\sqrt{n}$

$\Rightarrow n^2 + n^2\sqrt{n} \leq 2n^2\sqrt{n}$

$c = 2$, $f(n) = n^2\sqrt{n}$

Hence, for $\forall n \geq 1$, we have:- $T(n) \leq 2n^2\sqrt{n}$

$\Rightarrow T(n) = O(n^2\sqrt{n})$

$\neq O(n^2)$

(As:- $n^2\sqrt{n} \not\geq n^2(1)$ (Asymptotically))

So, answer:- false

c) $n^2(1+\sqrt{n}) = O(n^2 \log n)$

$n^2(1+\sqrt{n}) \leq n^2(\sqrt{n}+\sqrt{n})$

$\Rightarrow n^2 + n^2\sqrt{n} \leq 2n^2\sqrt{n}$

$c = 2$ & $f(n) = n^2\sqrt{n}$

Hence, for $\forall n \geq 1$, we have,

$T(n) \leq 2n^2\sqrt{n}$

$\Rightarrow T(n) = O(n^2\sqrt{n})$

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

(As $n^2\sqrt{n} \geq n^2(\log n)$ (Asymptotically))

Answer:- False

d) $3n^2 + \sqrt{n} = O(n + n\sqrt{n} + \sqrt{n})$

$3n^2 + \sqrt{n} \leq 3n^2 + n^2$

$\Rightarrow 3n^2 + \sqrt{n} \leq 4n^2$

$c = 2$ & $f(n) = n^2$

Hence, for $\forall n \geq 1$, we have $T(n) \leq 4n^2$

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

We have been given $O(n + n\sqrt{n} + \sqrt{n})$

$= O(n\sqrt{n})$ (Ignoring the lower order constants)

Now, $T(n) = O(n \cdot n)$

$\neq O(n \cdot \sqrt{n})$

As, $(n^2(n \cdot n) \geq n \cdot \sqrt{n}$ (Asymptotically))

Hence, Answer is false

e) $\sqrt{n} \log n = O(n)$

We know by, Growth rate of function:-

$\log n < \sqrt{n}$ (Asymptotically)

$\Rightarrow \sqrt{n} \log n < \sqrt{n} \cdot \sqrt{n}$

$\Rightarrow \sqrt{n} \log n < (1)n$

So, $c = 1.8$ & $f(n) = n$

Hence $\forall n \geq 1$, we have $T(n) \leq n$

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

Hence, Answer is true.

f) $\lg n \in O(n)$

from the growth rate of function:-

$1 < \lg n < \sqrt{n} < n < n \log n < n^2 < n^3 \cdots$

So, $\lg n = O(\lg n)$

$= O(n)$ (As $\lg n < n$ (Asymptotically))

$\Rightarrow$ Hence, answer is true

g) $n \in O(n \lg n)$
from the series of growth rate of fun', we have:-
$1 \leq \log n$
$\Rightarrow c(1) \leq c(n) \leq c(n) \log n$
$\because c=1$, & $f(n) = n \log n$
Hence, $\forall n \geq 1$, we have $T(n) \leq n \log n$
$\Rightarrow T(n) = O(n \lg n)$

Answer:- True

h) $n \lg n \in O(n^2)$
from bst (f), we have:-
$\lg n \leq n$
$\Rightarrow c(n) \lg n \leq c(n)(n)$
$\Rightarrow n \lg n \leq n^2$
$\because c=1$ & $f(n) = n^2$
Hence, $\forall n \geq 1$, we have, $T(n) \leq n^2$
$\Rightarrow T(n) = O(n^2)$

Answer: True

i) $2^n \in \Omega(6^{\ln n})$
we have:- $6^{\ln n} = n^{\ln 6} = n^{1.79}$
$\because$, from the series of growth rate of func. we have:-
$n^{1.79} \leq 2^n$
$\Rightarrow 2^n \geq n^{1.79}$ $\because c=1, f(n) = n^{1.79}$
$\because \forall n \geq 1$, we have:- $T(n) \geq n^{1.79}$
$\Rightarrow T(n) = \Omega(n^{1.79})$
$= \Omega(6^{\ln n})$.

Answer: True

j) $\lg^3 n \in O(n^{1/2})$
from the series of growth rate of func., we have:-
$\lg n \leq \sqrt{n}$
$\Rightarrow (\lg n)(\lg n)(\lg n) \leq (\sqrt{n})(\sqrt{n})(\sqrt{n})$
$\Rightarrow \lg^3 n < n^{3/2}$ $\rightarrow c=1$ & $f(n) = n^{1/2}$
$\Rightarrow \boxed{\lg^3 n < n^{1/2}}$ (strictly greater)
Hence, $\forall n \geq 1$ we have $T(n) < n^{1/2}$
$\Rightarrow T(n) = O(n^{3/2})$
$= O(n^{1/2})$ (strictly greater)

Answer: True

Question-05

For each of the following pairs of function $f(n)$ & $g(n)$, determine whether $f(n) = O(g(n))$ or $f(n) = \Omega(g(n))$ or $f(n) = \Theta(g(n))$.

a) $f(n) = \sqrt{n}$, $g(n) = \log(n+3)$

We know, $\sqrt{n} \geq \log n$, $\forall n \geq 1$

$\Rightarrow \sqrt{n} \geq (1) \log(n+3) \Rightarrow c = 1$ & $f(n) = \log(n+3)$

So, $\forall n \geq 1$, we have, $\boxed{f(n) = \Omega g(n)}$

b) $f(n) = n\sqrt{n}$, $g(n) = n^2 - n$

Time complexity for $f(n) = \Theta(n\sqrt{n}) = O(n\sqrt{n})$

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

Now, $\sqrt{n} < 1$ → from the series of growth of fun^c.

$\Rightarrow \sqrt{n} \leq n$

$\Rightarrow \boxed{n\sqrt{n} \leq n^2}$

$\Rightarrow n\sqrt{n} \leq c \cdot (n^2 - n)$

$\Rightarrow f(n) \leq c \cdot (g(n))$

$\Rightarrow f(n) = O(g(n))$ → (answer)

c) $f(n) = 2^n - n^2$, $g(n) = n^4 + n^2$

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

$T(n)$ for $g(n) = O(n^4)$

So, from the growth of rate of fun^c.:-

$2^n \geq c \cdot n^4$

$\Rightarrow f(n) \geq c \cdot g(n)$

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

d) $f(n) = n^2 + 3n + 4$, $g(n) = 6n^2$

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

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

from the series of growth rate of fun^c.:-

$\Rightarrow n^2 \leq c \cdot n^2$

$\Rightarrow n^2 = n^2$

OR $f(n) = \Theta g(n)$ → answer.

e) $f(n) = n + n\sqrt{n}$, $g(n) = 4n \log(n^2+1)$

$f(n) = n + n\sqrt{n}$

$T(n)$ for $f(n) = O(n\sqrt{n})$

$g(n) = 4n\log(n^2+1)$

$T(n)$ for $g(n) = O(n \log n)$

from the series of growth rate of fun^n, we have,

$\log n \leq \sqrt{n}$

$\Rightarrow \sqrt{n} \geq \log n$

$\Rightarrow n\sqrt{n} \geq n \log n$

$\Rightarrow f(n) \geq c \cdot g(n)$

So, $f(n) = \Omega g(n)$ → answer

Question-06

Prove the following statements:-

a) If $f_1(n) = \Omega(g_1(n))$ & $f_2(n) = \Omega(g_2(n))$ then $f_1(n) + f_2(n) = \Omega(g_1(n) + g_2(n))$

We have, $f_1(n) = \Omega(g_1(n))$

$f_2(n) = \Omega(g_2(n))$

We can write it as:-

$0 \leq c_1 g_1(n) \leq f_1(n)$ — ①

& $0 \leq c_2 g_2(n) \leq f_2(n)$ — ②

Adding ① & ②, we get

$0 \leq c_1 g_1(n) + c_2 g_2(n) \leq f_1(n) + f_2(n)$

$\Rightarrow 0 \leq (g_1(n) + g_2(n)) c_3 \leq f_1(n) + f_2(n)$

$\Rightarrow f_1(n) + f_2(n) = \Omega(g_1(n) + g_2(n))$

Hence Proved.

b) If $f_1(n) = \Omega(g_1(n))$ & $f_2(n) = \Omega(g_2(n))$ then $f_1(n) + f_2(n) = \Omega(\min(g_1(n) + g_2(n)))$

Given: $f_1(n) = \Omega(g_1(n))$

$\Rightarrow 0 \leq c_1 g_1(n) \leq f_1(n)$ — ①

$f_2(n) = \Omega(g_2(n))$

$\Rightarrow 0 \leq c_2 g_2(n) \leq f_2(n)$ — ②

Adding ① & ② we get:-

$0 \leq c_1 g_1(n) + c_2 g_2(n) \leq f_1(n) + f_2(n)$

$\Rightarrow 0 \leq c_3 [g_1(n) + g_2(n)] \leq f_1(n) + f_2(n)$

$\Rightarrow 0 < c(g_1(n)+g_2(n)) \leq f_1(n)+f_2(n)$

Here, c is the minimum value s.t. it is less than $f_1(n)+f_2(n)$

It is happening as $c_3 > c$, but still less than $f_1(n)+f_2(n)$

So, $c(g_1(n)+g_2(n)) \leq f_1(n)+f_2(n)$

Hence, $f_1(n)+f_2(n) = \Omega(\min(g_1(n)+g_2(n)))$

↳ proved.

c) If $f_1(n) = O(g_1(n))$ & $f_2(n) = O(g_2(n))$ then $f_1(n) \cdot f_2(n) = O(g_1(n) \cdot g_2(n))$

Given:- $f_1(n) = O(g_1(n))$

$\Rightarrow 0 \leq f_1(n) \leq c_1(g_1(n))$ — ①

Also, $f_2(n) = O(g_2(n))$

$\Rightarrow 0 \leq f_2(n) \leq c_2(g_2(n))$ — ②

Multiplying ① & ②, we get:-

$0 \leq f_1(n) \cdot f_2(n) \leq \boxed{c_1 c_2}(g_1(n)) \cdot (g_2(n))$

$\Rightarrow 0 \leq f_1(n) \cdot f_2(n) \leq c_3(g_1(n)) \cdot (g_2(n))$

Hence, $f_1(n) \cdot f_2(n) = O(g_1(n) \cdot g_2(n))$

↳ from the defn of Big 'O'.

d) If $f(n) = O(g(n))$, then $f(n)^k = O(g(n)^k)$

Given:- $f(n) = O(g(n))$

$\Rightarrow 0 \leq f(n) \leq c_1(g(n))$

Multiplying the power k to inequalities, we get

$0^k \leq f(n)^k \leq c_1^k g(n)^k$

$\Rightarrow 0 \leq f(n)^k \leq c_2 g(n)^k$

So, $f(n)^k = O(g(n)^k)$ - Proved.

e) If $f_1(n) = O(g_1(n))$, $f_2(n) = O(g_2(n))$, ---, $f_k(n) = O(g_k(n))$ where $k \geq 1$ then prove that $\sum_{i=1}^{k} f_i(n) = O(\max_{1 \leq i \leq k}(g_i(n)))$

We have:- $f_1(n) = O(g_1(n))$

$f_2(n) = O(g_2(n))$

$f_k(n) = O(g_k(n))$

So, we can write:-

$0 \leq c_1[g_1(n)] \leq f_1(n) \leq c_1[g_1(n)]$

$0 \leq c_k[g_k(n)] \leq f_k(n) \leq c_k[g_k(n)]$

Note:- Here $c_1', c_2', \dots c_k'$ can be max as of $f_k(n), f_1(n) \le c_k' g_k(n) \le c_1' g_1(n) \le c_k' \max g_k(n)$

So, adding all the above eqn:-

$0 \le \min[g_1(n) + g_2(n) + \dots g_k(n)] \le f_1(n) + \dots f_k(n)$

$\le \max[g_1(n) + g_2(n) + \dots g_k(n)]$

$\Rightarrow \sum_{i=1}^{k} f_i(n) \in O(\max_{1 \le i \le k}(g_i(n)))$ ↳ Proved.

Question-07

Given $f(n) = (n+a)^b$ for any real constants a & b, where $b > 0$. Prove $f(n) \in \Theta(n^b)$.

$(n+a)^b = n^b + c_1 n^{b-1} a + \dots \in \Theta(n^b)$ → we need to prove this ↳ ①

from eqn ①, we can write, $0 \le c_1 n^b \le (n+a)^b \le c_2 n^b$ $\forall n \ge n_0$

$n + a \ge n - |a|$

$n + a \ge n/2, |n| \le n/2$

Let us consider:- $f(n) = (n+a)^b$ & $g(n) = n^b$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} \Rightarrow \lim_{n\to\infty} \frac{(n+a)^b}{n^b} = \lim_{n\to\infty} \frac{n^b(1+a/n)^b}{n^b}$$

$$= \lim_{n\to\infty} (1 + a/n)^b$$

$= 1$

$\Rightarrow f(n) \in O(g(n))$ — (II)

Now,

$$\lim_{n\to\infty} \frac{g(n)}{f(n)} = \lim_{n\to\infty} \frac{n^b}{(n+a)^b} = \lim_{n\to\infty} \frac{n^b}{n^b(1+a/n)^b} = 1/1^b = 1$$

$\Rightarrow f(n) = \Omega(g(n))$ — (III)

Both (II) & (III) gives that →

$f(n) = \Theta(g(n)) \Rightarrow (n+a)^b \in \Theta(n^b)$.

Question-08

Prove that $\lg n = O(\sqrt{n})$, however $\sqrt{n} \ne O(\lg n)$

We know, $\lg n \le n$ (from mathematics)

$\Rightarrow \lg \sqrt{n} \le \sqrt{n}$

$\Rightarrow 2 \lg \sqrt{n} \le 2\sqrt{n}$

$\Rightarrow \lg (\sqrt{n})^2 \le 2\sqrt{n}$

Question-10

2) $\lg n < 2\sqrt{n}$

$c = 2$, $f(n) = \sqrt{n}$

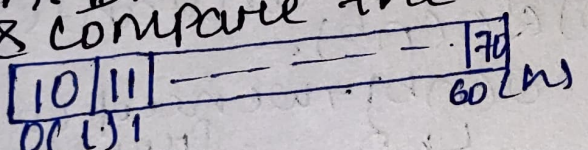
$\forall n \geq 1$, we have $T(n) \leq 2\sqrt{n}$

2) $T(n) = O(\sqrt{n})$

Hence Proved.

Question-09

Let $A[1 \ldots 60] = [10, 11, \ldots, 70]$. How many comparisons are performed by algorithm binary search(A) when searching for the elements 10, 40, 50, 70 respectively? Explain in each cases with detailed execution. Draw the recursion tree for each of the cases & compare the stack space used.



Further Answer as in page no. 5 & 6.

Question-10

Suppose you're given with a problem P with input size n and 3 algorithms A, B and C to choose from, in order to solve P. The algorithms operate as follows:

- Algorithm A solves P by dividing it into five sub-problems of half the size, recursively solving each sub-problem & then combining the soln in linear time.
- Algorithm B solves P by recursively solving 2 sub-problems of size n-1 & then combining the soln in constant time.
- Algorithm C solves P by dividing it into nine sub-problems each of size n/3, recursively solving each sub-problem & then combining the solns in $O(n^2)$ times.

Compare the running times of these algorithm in big O notation. Which one would you choose?

* Recurrence relation for algo. A is represented by using Master's theorem:-

$T(n) = 5T(n/2) + n, \quad T(1) = 1$

In the above rel^n $(n/2)$ is taken because each time in sub problem input becomes half.

Then as per the Master's thm., $a = 5, b = 2$ & $f(n) = n$

So, T(n) / running time for Algorithm A $= O(n^{\log_b a})$

$= O(n^{\log_2 5})$

* Recurrence rel^n for algo. B is:-

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

$T(1) = 1$

Again the next recurring terms T(3) & T(4) for algo. B will be:-

$T(3) = 2T(2) + (2)(2)\ T(1) + c$

$T(4) = 2T(3) + (2)(2)(2)\ T(2) + (2)(2) T(1) + c$

Generalizing, the above recurring relation, it should be noted that the relation is subdivided & deepens in the multiple of 2^n.

Therefore, the running time of algo. B is $O(2^n)$.

* Recurrence rel^n for algo. C as per the Masters thm. will be:-

$T(n) = 9T(n/3) + (n^2), \quad T(1) = 1$

According to Master's thm., $a=9, b=3, k=\log_3^9 = 2$

$f(n) = n^2$

So, $T(n) = O(n^2 \log n)$

Since, algo. C linearly divides the problem into 9 sub problems of size $(n/3)$ & takes running time in terms of $(\log_3 n)$ which consumes less memory space in comparison to the recursive approach.

So, C will be an appropriate algo. for implementation.

Question-11

```
function(int n)
{ if(n==1) → 1
    return 1; → 1
  else
   function(n/3); function(n/3); function(n/3);
   for(i=1; i<2n; i++) → n+1                    → 3T(n/3)
     x = x+1 → n
}
```

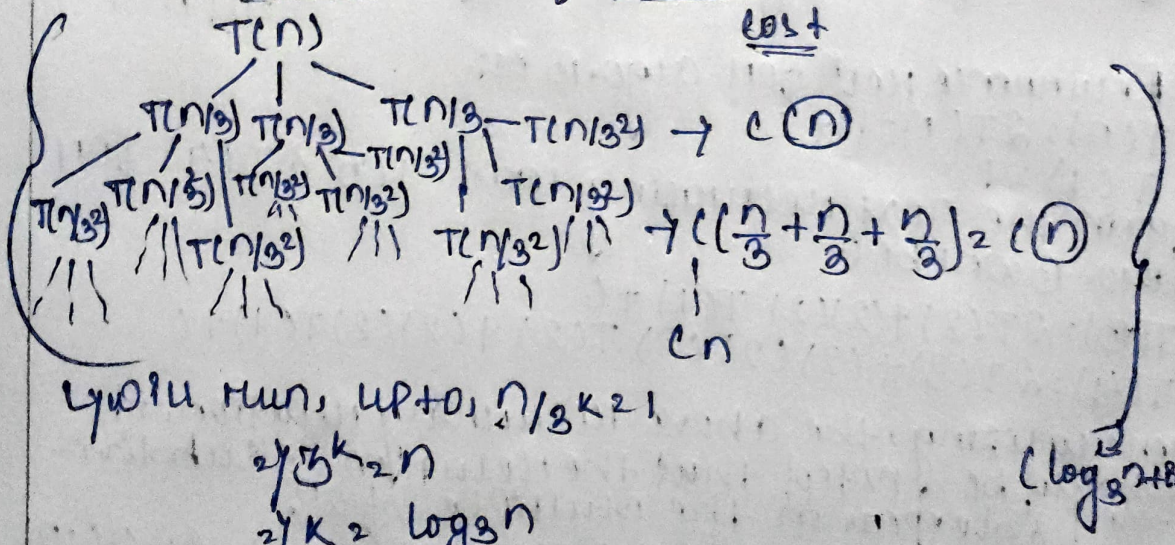
3

Find the time and space complexity of the given algorithm.

Time complexity: $T(n) = 1+1+3T(n/3)+n+1+n$

$= 3T(n/3) + O(n)$

Now recursive tree for the function:-



Upto run, upto, $n/3^k = 1$

$n = 3^k$

$k = \log_3 n$

height of the tree.

Space complexity:- As the T(n) is in recursive form
→ So, the space complexity is the max. height of the tree
→ As the tree is in the same level, so the maximum height of the tree = $\log_3 n$
Also there some constant variables in the algo., not any array of variables space required.
So, $S(n) = \log_3 n + O(1)$
Space complexity = $O(\log_3 n)$

Question-12

```
Void function(int n){
        Temp=1;
        Repeat
          for i=1 to n
            temp = temp+1
            n = n/2;
        until(n<=1)
```

3 find the time & space complexity.

```
Void function(int n) {  →①
temp=1;
for (int i=1; i<=n, n<=1; n=n/2)  → log2n
      temp = temp+1;  → log2n - 1
```

So, $T(n) = 1 + \log_2 n + \log_2 n - 1 = 2\log_2 n - O(\log_2 n)$
Time complexity = $O(\log_2 n)$
Space complexity = $O(1)$ → As only constant no. of variable required & no variables requires an array & space.

Question-13:-

```
int function(int n){
if (n<2)  -①
    return 1; -①
else -①   return(function(floor(sqrt(n)))+1); -T(√n)
}
```

find the time comp. & space comp.

Any $T(n) = \begin{cases} 1, & n<2 \\ T(\sqrt{n})+1, & n \geq 2 \end{cases}$

Recursive trace for time complexity:-

	cost
$T(n)$	
$\downarrow$	
$T(\sqrt{n})$	→ ①
$\downarrow$	
$T(n^{1/2^2})$	→ ①
$\downarrow$	
$T(n^{1/2^3})$	→ ①
!	
$T(n^{1/2^k})$	→ ①

k times

$n^{1/2^k} = 2$.

$\Rightarrow 1/2^k \log_2 n = 1$

$\Rightarrow 2^k = \log_2 n$

$\Rightarrow k = \log_2 \log_2 n$

So, time complexity $= c * k$

$= O(\log \log n)$

Now SC = S(n) = height of the tree.

$= O(\log \log n)$

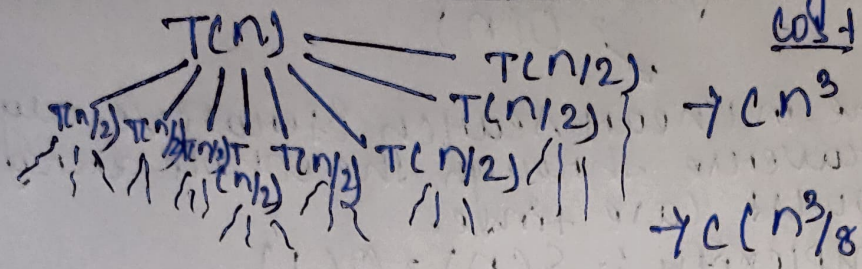
Question-14

```
void function(int n){
if(n == 1)
    return 1;
else for(i=1; i<28; i++)
        function(n/2);
for(i=1; i<n^3; i++)
    count = count+1;
}
```

Find the TC & SC.

$T(n) = 1+1+8T(n/2)+O(n^3)$

Recursive trace for time complexity:-



$[(n/2^k)] = 1$ (Termination point) $\Rightarrow k = \log_2 n$

$TC = c(n^3) * \log_2 n$

$= O(n^3 \log_2 n)$

Space complexity:-

$S(n) \rightarrow$ max. height of the recursive tree

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

(Constant no. of variables).

$\Rightarrow$ space complexity $(S(n)) = O(\log n)$

Question-15

The following pseudocode performs linear search on an array of size n to find the presence of an element e.

```
Linearsearch(A, n, el)
1. for i = 1 to n do
2. if A[i] = el then
3. return i
4. return NIL
```

Write the recursive version of the linear search Algorithm & compare the time & space complexity with the iterative version.

Recursive Algorithm:-

```
int LinearSearch(A, n, el) {
  n--;
if(n<0)
    return -1;
 if(A[n] == el)
     return n;
```

Return linearsearch(a, n, e)
Time complexity :- T(n) → n+1+n+1+1
= O(n)

As the in linear search iterative algorithm is used it will traverse through the whole array in the worst case time.

Space complexity :- S(n) = O(1)
→ fixed/constant variables
→ No extra array space is req, hence O(1)

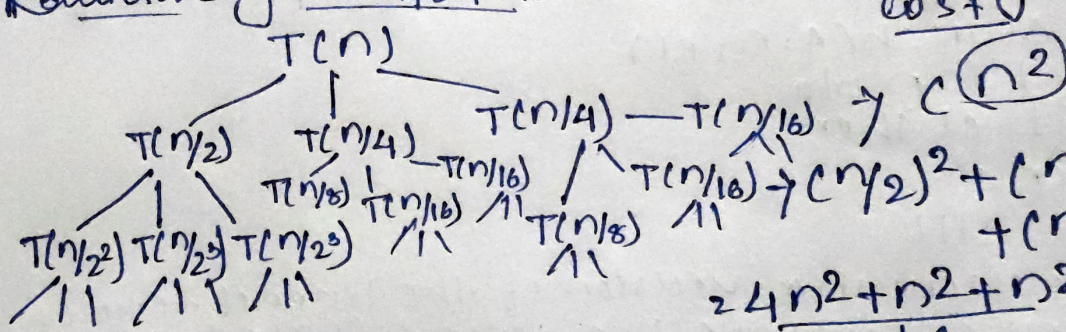
Question-16

Solve the following recurrence using any of the suitable methods. If no soln is possible justify using proper reasoning.

a. $T(n) = \begin{cases} 1 & \text{if } n = 2 \\ 1 & \text{if } n = 4 \\ T(n/2) + 2T(n/4) + \Theta(n^2) & \text{if } n > 4 \end{cases}$

where n is assumed to be a power of 2.

Recursive tree for the time complexity:-



$$= \frac{4n^2 + n^2 + n^2}{16}$$

$$= \frac{3}{8}n^2$$

$$\rightarrow \frac{n^2}{16} + \frac{n^2}{64} + \frac{n^2}{64} + \frac{n^2}{64} + \cdots$$

$$+ \frac{n^2}{256} + \frac{n^2}{256} + \frac{n^2}{64} + \frac{n^2}{256} + \frac{n^2}{256}$$

$$= \frac{16+4+4+4+1+1+4+1+1}{256} n^2 = \frac{36}{64} n^2 = \frac{9}{64} n^2 = \left(\frac{3}{8}\right)^2 n^2$$

So, $T(n) = n^2 + \frac{3}{8}n^2 + (\frac{3}{8})^2 n^2 + \cdots$ h times (where h is the height of the tree)

$\Rightarrow T(n) \leq n^2 (1 + \frac{3}{8} + (\frac{3}{8})^2 + \cdots \infty \text{ times})$

$\Rightarrow T(n) \leq n^2 \left(\frac{1}{1 - 3/8}\right)$

$\Rightarrow \boxed{T(n) = O(n^2)}$ → Ans

b) $T(n) = n^{1/3} T(n^{2/3}) + \Theta(n)$

We have, $T(n) = n^{1/3} T(n^{2/3}) + \Theta(n)$

$= n^{1/3} T(1/n^{-2/3}) + \Theta(n)$

$= n^{1/3} T(n/n^{-2/3+1}) + \Theta(n)$

$= n^{1/3} T(n/n^{1/3}) + \Theta(n)$

Using Masters thm.

$\boxed{a = n^{1/3}, b = n^{1/3}}$ $f(n) = \Theta(n)$

So, acc. to Masters thm.,

$n^{\log_b^a} = n^{\log_{n^{1/3}}^{n^{1/3}}} = \Theta(n) = \Theta f(n)$

So, acc. to Case-II $\Rightarrow T(n) = \Theta(n^{\log_{n^{1/3}}^{n^{1/3}}} \cdot \log n)$

$= \Theta(n \cdot \log n)$ — ans

c) $T(n) = \sqrt{n}\, T(\sqrt{n}) + \log n$

Using recursive tree, for time complexity:-

$T(n)$	cost
$(n)^{1/2} T(n^{1/2})$	$\log n$
$(n)^{(1/2)^2} T(n^{1/2})^{1/2}$	$1/2 \log n$
$(n)^{(1/2)^3} T(n^{(1/2)^3})$	$(1/2)^2 \log n$
$(n)^{(1/2)^k} T(n^{(1/2)^k})$	$(1/2)^{k-1} \log n$

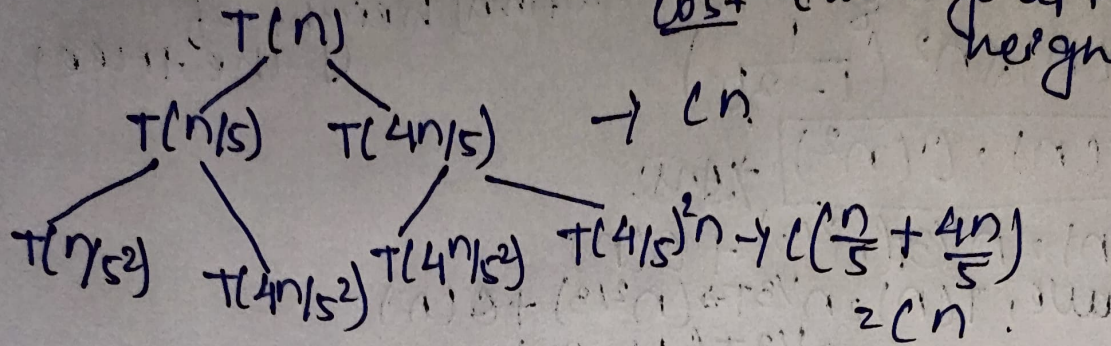
So, $T(n) = \log n (1 + 1/2 + (1/2)^2 + (1/2)^3 \cdots (1/2)^{k-1})$

$= \log n (1 + 1/2 + \cdots \infty)$

$\Rightarrow T(n) = O(\log n)$ → ans.

d) $T(n) = T(n/5) + T(4n/5) + \Theta(n)$

Recursive tree for time comp.:- (cost (will go up to height))



Max. height $= \log_{5/4} n$ ($5/4 < 5/1$).

$TC = cn * \log_{5/4} n$

$= O(n \log_{5/4} n)$ → ans

e) $T(n) = 3T(n/4) + cn^2$

Using master's theo., we have $a = 3, b = 4$.

So, $k \log_b^a = n^{\log_4^3}$

So, $f(n) = n^2 = \Omega(n^{\log_4^{3+\varepsilon}})$, where, $\varepsilon = 1.3$, $\varepsilon > 0$.

Now, $af(n/b) \leq cf(n)$ → Regularity condition

$\Rightarrow 3f(n/4) \leq c(n^2)$

$\Rightarrow 3(n^2/16) \leq cn^2$ $c = 4/16 < 1$, & large value of n.

then $T(n) = \Theta(f(n)) = \Theta(n^2)$

f) $T(n) = \begin{cases} 8 & \text{for } n = 1 \\ 3T(n-1) - 15 & \text{for } n > 1 \end{cases}$

We have, $T(n) = 3T(n-1) - 15$, $n > 1$

$a = 3, b = 1, f(n) = O(1), a > 0, b > 0$

$= O(n^0), k > 0$

Case-III If $a > 1$; $T(n) = O(3^n * O(n^0))$

$= O(3^n * n^0)$

$= O(3^n)$ → ans.

g) $T(n) = \begin{cases} 5 & \text{for } n \geq 1 \\ 2T(n-1)+3n+1 & \text{for } n > 1 \end{cases}$

We have :- $T(n) = 2T(n-1)+3n+1,\ n>1$

$a=2, b=1, f(n)=O(n)$

Case-III :- If $a>1$; $T(n) = O(2^n * n)$

$= O(n2^n)$ → ans

h) $T(n) = n/n-5\ T(n-1)+1$

We have, by substitution method :-

$T(n) = n/n-5\ T(n-1)+1$

→ $T(n-1) = \frac{n-1}{n-6} T(n-2)+1$

→ $T(n-2) = \frac{n-2}{n-7} T(n-3)+1$

$\therefore T(n) = \frac{n}{n-5} \cdot \frac{(n-1)}{(n-6)} \cdot \frac{(n-2)}{(n-7)} T(n-3)+1+1+1$

$= \frac{(n)(n-1)\cdots(n-(k-1))}{(n-5)(n-6)\cdots(n-k+4)} T(n-k)+k$

If $k=n$, then $T(n) = \frac{n!}{(n-5)!} + n$

$= n(n-1)(n-2)(n-3)(n-4)+n$

→ $T(n) = O(n^5)$ — ans.

i) $T(n) = T(\log n) + \log n$

using master theorem, $a=1, b=1, f(n) = \log n$

Now, $n^{\log_b a} = n^{\log_1 1} = n^0$ (dividing func^n)

$\therefore$ case-1 :- $f(n) \leq C(n^{\log_b a} - \epsilon)$ $\epsilon < 0$,

$\therefore T(n) = O(\log^a_b)$

$= O(n)$ — ans

j) $T(n) = T(n^{1/4}) + 1$

using master's theorem,

we have, $T(n) = (1)\, T(n^{1/4}) + 1$

$= (1)\, T(n^{-(-1/4)}) + 1$

$= 1\,T(n^{1/4}) + 1$

$= 1\,T(n/n^{3/4}) + 1$

$\therefore a = 1,\ b = n^{3/4},\ f(n) = O(1)$

Case-II :- $f(n) = O(1)$

$n^{\log_b^a} = n^{\log_{3/4}^1} = n^0 = 1 = O(1)$

$\therefore f(n) = \Theta(n^{\log_b^a})$

Then, $T(n) = \Theta(n^{\log_b^a} * \log n)$

$= \Theta(1 * \log n) = \Theta(\log n)$ Ans

L) $T(n) = T(3n/4) + 1/\sqrt{n}$

$T(n) = 1\,T(n/(4/3)) + 1/\sqrt{n}$

using master's thm :- $T(n) = a(T(n/b)) + f(n)$

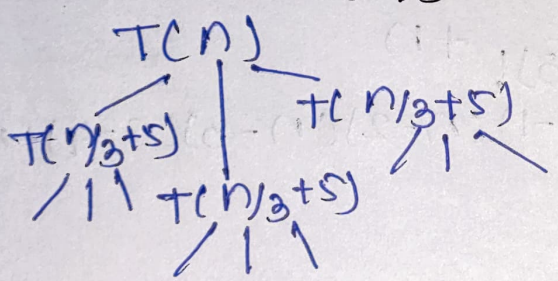
$\therefore a = 1,\ b = 4/3,\ f(n) = 1/\sqrt{n} = n^{-1/2}$

Now, $n^{\log_b^a} = n^{\log_{4/3}^1} = n^0$

$\therefore f(n) \leq O(n^{\log_b^a}) \rightarrow \therefore T(n) = \Theta(n^{\log_b^a})$

$= \Theta(1)$ — ans

M) $T(n) = \begin{cases} 1, & \text{for } n = 1 \\ 3T(n/3+5) + n/2 & \text{for } n > 1 \end{cases}$



cost

$\rightarrow n/2$

$\rightarrow n/6 + n/6 + n/6 = n/2$

$\hookrightarrow$ k times.

Suppose at $T(n/3^k + 5)$ the tree will stop

$\therefore T(n/3^k + 5) = T(1) = 1$

$\Rightarrow n/3^k = -4$ let -4 be c $\Rightarrow$ where c is constant

$\Rightarrow n = c\,3^k$

$\Rightarrow k = c \log_3 n \rightarrow$ where k is the height of tree

$\therefore TC = \text{height} * \text{cost} = c * \log_3 n * n/2$

$= (n/2 \log_3 n)$

Hence, $TC = O(n/2 \log_3 n)$

$= O(n \log n)$

n) $T(n) = \begin{cases} 1 & n = 1 \\ 2T(n/2) + \frac{n}{\log\log n} & \text{for } n > 1 \end{cases}$

using master's thm.,

$T(n) = aT(n/b) + f(n)$.

Here, $a = 2, b = 2, f(n) = n/\log\log n$.

Now, $n^{\log_b a} = n^{\log_2 2} = n$

Now, case I of master's thm.:-

$f(n) \leq O(n^{\log_2 2}) \Rightarrow T(n) = O(n^{\log_2 2})$

$= O(n)$

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

using master's thm,

$T(n) = aT(n-b) + f(n)$

where $a > 0, b > 0$

$f(n) = O(n^k)$ $k \geq 0$

Case I:-

$a = 1$, $T(n) = O(n \times f(n))$, where $f(n) = 2\log n$

$= O(n * 2\log(n))$

$= O(n \log(n))$ — ans.

p) $T(n) = 2T(n/2) + n^2(1 + \sin n)$

using master's thm.

$T(n) = aT(n/b) + f(n)$

Here, $a = 2, b = 2, f(n) = n^2(1 + \sin n)$

Now, $n^{\log_b a} = n^{\log_2 2} = (n)$

Case-III

$f(n) \Rightarrow \Omega(n^{\log_b a})$

$\Rightarrow$ It should satisfy the regularity condition

i.e., $af(n/b) \leq cf(n)$

$\Rightarrow 2 \cdot (n^2/4 (\sin n/2 + 1)) \leq c \cdot (n^2)(1 + \sin n)$

$\Rightarrow n^2/2 (1 + \sin n/2) \leq (c)(n^2/2)(1 + \sin n)$

where $c = 1/2 < 1$

Hence, as the regularity condition is satisfi

$T(n) = \Theta(f(n))$

$= \Theta(n^2(1+\sin n))$

$= \Theta(n^2 + n^2 \sin n)$

$= \Theta(n^2 \sin n)$ ans

— X —